p–A Small C Preprocessor

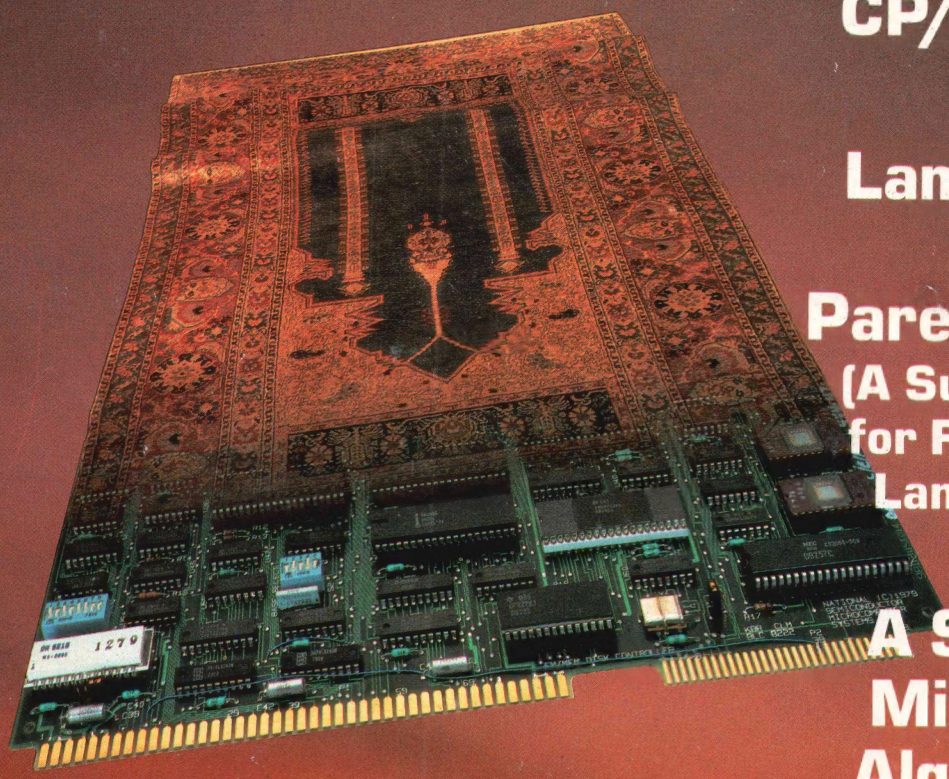
SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS

# Dr. Dobb's Journal

#93 July 1984

$2.95 (3.50 Canada)

## Resident System Extensions under CP/M Plus



## Languages and Parentheses (A Suggestion for Forth-like Languages)

## A Simple Minimax Algorithm

0 38351 16562 07

# P E R F O R M A N C E

By Design

**The recent Datapro Microcomputer User Survey reported a 3.8 overall user satisfaction rating out of a possible 4.0 for Sage Computers.**


Sure, we like to read about ourselves scoring high marks in market studies. Our users do also. We appreciate the positive comments written about us by writers and editors around the world. But, as much as we enjoy the reports, it doesn't really surprise us.

We've designed performance into every computer system we manufacture. Not just speed, but flexibility, functionality and reliability. Sage has been building high performance 68000 multi-user systems longer than anyone, and we know that designing performance into our product requires time, attention to detail and a non-compromising attitude of doing things right.

Sage systems are available with nine different operating systems, 23 languages and over 300 application programs in 50 different categories. All systems come with a 90-day warranty, extendable to 3 years. And we have hundreds of dealers worldwide.


If you would like to know more about Sage and our Sage II and IV microcomputer systems, call or write today for your free copy of the 28-page Sage Product Catalog. It offers all you need to know about Sage, and how we design performance into every product we sell.

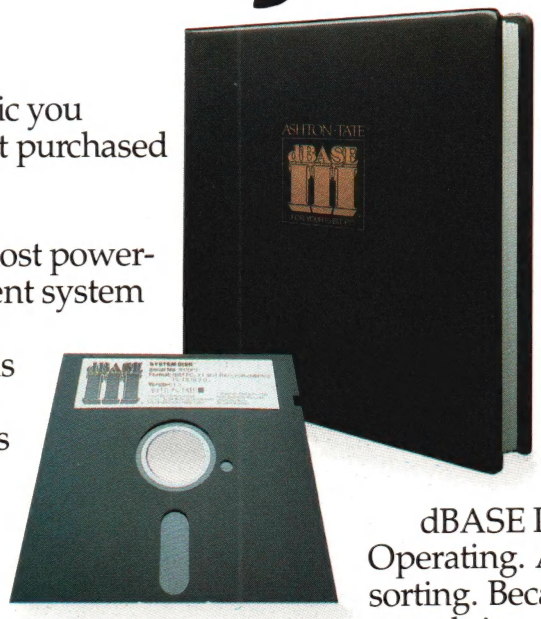
***Reno: 702-322-6868***
***Dallas: 214-392-7070***
***Boston: 617-229-6868***

**SAGE** COMPUTER

***Reno:*** 4905 Energy Way, Reno, Nevada 89502 (TWX 910-395-6073 SAGERNO)
***Dallas:*** 14755 Preston Rd., Suite 600, Dallas, TX 75240
***Boston:*** 15 New England Exec. Park, Suite 120, Burlington, MA 01803

Circle **no. 62** on reader service card.

# More power to you.




Remember the magic you expected when you first purchased a PC?

It's here.

dBASE III™ is the most powerful database management system ever created for 16-bit microcomputers. It pulls every ounce of energy out of your PC and puts it to work.

On top of that, it's fast and it's easy.

## You've never seen anything like it.

dBASE III can handle over a billion records per file, limited only by your computer system. You can have up to ten files open, for sophisticated applications programs.

When you have two related files, information in one can be accessed based upon data in the other.

dBASE III now handles procedures, parameter passing and automatic variables. You can include up to 32 procedures in a single file. With lightning speed. Because once a file is opened, it stays open. And procedures are accessed directly.

## Easier than ever.

dBASE III uses powerful yet simple commands that are the next best thing to speaking English.

If you're unsure of a command, HELP will tell you what to ask for.

If you don't know what command comes next, a command assistant does. All you have to know is what you want it to do.

Our new tutorial/manual will have you entering and viewing data in minutes rather than reading for hours.

And to make matters easier, you get a full screen report setup for simple information access.

## Faster than no time at all.

dBASE III isn't just fast. It's ultra-fast. Operating. And sorting. Even faster, is no sorting. Because dBASE III keeps your records in order, so you really don't have to sort anything. Unless you want to. Then watch out!

## What about dBASE II®?

It's still the world's best database management system for 8-bit computers. And it's still the industry standard for accounting, educational, scientific, financial, business and personal applications.

## Tap into our power.

For the name of your nearest authorized dBASE III dealer, contact Ashton-Tate, 10150 West Jefferson Boulevard, Culver City, CA 90230. (800) 437-4329, ext. 333. In Colorado, (303) 799-4900.

ASHTON·TATE™

# Dr. Dobb's Journal

**Editorial**

**Editor-in-Chief** *Michael Swaine*
**Editor** *Reynold Wiggins*
**Managing Editor** *Randy Sutherland*
**Contributing Editors** *Robert Blum, Dave Cortesi, Ray Duncan, Anthony Skjellum, Michael Wiesenberg*
**Copy Editors** *Polly Koch, Cindy Martin*
**Typesetter** *Jean Aring*

**Production**

**Design Director** *Fred Fehlau*
**Art Director** *Shelley Rae Doeden*
**Production Manager** *Detta Penna*
**Production Assistant** *Alida Hinton*
**Cover Photo** *Tom Upton*

**Advertising**

**Advertising Director** *Susan Cohen Strange*
**Advertising Sales** *Alice Hinton, Walter Andrzejewski*

**Circulation**

**Circulation and Promotions Director** *Beatrice Blatteis*
**Fulfillment Manager** *Stephanie Barber*
**Direct Response Coordinator** *Maureen Snee*
**Promotions Coordinator** *Jane Sharninghouse*
**Single Copy Sales Coordinator** *Sally Brenton*
**Single Copy Sales** *Lorraine McLaughlin*
**Circulation Assistant** *Kathleen Boyd*

**M&T Publishing, Inc.**

**Publisher and Chairman of the Board** *Otmar Weber*
**Director** *C.F. von Quadt*
**President** *Laird Foshay*





**Dr. Dobb's Journal** (USPS 307690) is published monthly by M&T Publishing, Inc., 2464 Embarcadero Way, Palo Alto, CA 94303, (415) 424-0600. Second class postage paid at Palo Alto and at additional entry points.

Address correction requested. Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, 2464 Embarcadero Way, Palo Alto, CA 94303. **ISSN 0278-6508**

**Subscription Rates:** $25 per year within the United States, $44 for first class to Canada and Mexico, $62 for airmail to other countries. Payment must be in U.S. Dollars, drawn on a U.S. Bank.

**Contributing Subscribers:** Christine Bell, W.D. Rausch, DeWitt S. Brown, Burks A. Smith, Robert C. Luckey, Transdata Corp., Mark Ketter, Friden Mailing Equipment, Frank Lawyer, Rodney Black, Kenneth Drexler, Real Paquin, Ed Malin, John Saylor Jr., Ted A. Reuss III, InfoWorld, Stan Veit, Western Material Control, S.P. Kennedy, John Hatch, Richard Jorgensen, John Boak, Bill Spees, R.B. Sutton. **Lifetime Subscribers:** Michael S. Zick, F. Kirk.

**Foreign Distributors:** ASCII Publishing, Inc. (Japan), Computer Services (Australia), Computer Store (New Zealand), Computercollectief (Nederland), Homecomputer Vertriebs GMBH (West Germany), International Presse (West Germany), La Nacelle Bookstore (France), McGill's News Agency PTY LTD (Australia), Progresco (France).

**People's Computer Company**

**Dr. Dobb's Journal** is published by M&T Publishing, Inc. under license from People's Computer Company, Box E, Menlo Park, CA 94025, a non-profit, educational corporation.


## In This Issue

Our Resident Intern puzzled a few of months ago about the relative sparseness of CP/M Plus users. We're also surprised, in light of its greater flexibility and power than ordinary CP/M 2.2. Resident System Extensions are one example of this. This issue we have two pieces that deal with these handy routines. Garry Silvey of Digital Research has provided us with an article showing how they work and how to add them to your system. Bob Blum's CP/M Exchange column also received an RSX contribution from Mike Griswold that patches incompatibilities between CP/M Plus and 2.2 BIOS calls. Maybe this will give more folks incentive to explore what CP/M Plus can do for them. We plan to provide other material on CP/M Plus in the near future.

## The Doctor's Bulletin Board

In the April issue we intimated that an electronic bulletin board could be in *DDJ*'s future. Well, what we have in mind is a multi-user system that will allow lots of features to be included. Because it is intended, among other things, to improve the reader interface, we would like to hear your suggestions of desirable capabilities. An excellent way to make your recommendations is the Editorial Response card (we even pay the postage). If you have something you would really like the board to do, jot it on the card with the any other comments you wish to make and drop it in the mail. We can't promise *everything* will be included, but we want it to serve as many needs a possible.

## For Our Authors

Our latest version of the writer's guidelines is now available—expanded and accompanied by a current payment schedule. Just drop us a note with your name and address and we'll send them along to you. We are also adding ways to receive manuscripts. We can currently accept them via modem, MCI Mail, diskette in an increasing number of formats, and, of course, hard copy. You'll need to ask us about the particulars.

## This Month's Referees

*Dr. Dobb's Journal* regularly draws on the expertise of a Board of Referees for technical evaluation of material submitted for publication. In addition to remarks to the editors concerning accuracy and relevance on manuscripts, the referees often provide constructive comments for authors regarding clarity or completeness. Their remarks help prevent authors from exposing blindspots or misconceptions in print and help ensure that our readers receive clear and accurate information.

Numbering between forty and fifty referees, the referees include experts from diverse areas of the computer industry and the academic community. Because of space considerations, we can print a list of the entire board only a few times each year. Monthly, however, we do print the names of the referees who contributed their insights on material in that particular issue. Your humble editors must bear the burden of choosing how material ultimately appears, and we are grateful for the beneficial advice we receive.

The referees who contributed to this month's issue are:

Robert Blum, Contributing Editor, *DDJ*
David E. Cortesi, Contributing Editor, *DDJ*
James E. Hendrix, Office of Computing and Information Services, University of Mississippi
Richard G. Larson, Dept. of Mathematics, Statistics and Computer Science, University of Illinois at Chicago
William Ragsdale, President, Dorado Systems
Robert Smith

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS

# Dr. Dobb's Journal

July 1984
Volume 9, Issue 7

# CONTENTS

## ARTICLES



## DEPARTMENTS

# EDITORIAL

There were programmers long before the invention of the digital computer in the 1930s and 40s. A century earlier, Jacquard's loom took its weaving instructions on punched cards: the person who selected the cards to produce the desired pattern was an early programmer.

Software designers came later. What Mauchly and Eckert and Atanasoff and Berry and Von Neumann and Turing and Zuse invented under the shadow of World War II was not just a programmable device, but a general-purpose programmable device, and that distinction made possible (and necessary) software designers.

A software designer makes the computer do something qualitatively different from anything it has done before, supplies the general-purpose device with a new specific purpose.

Without that definition of purpose, documentation is moot. Documentation for specific applications such as word processors and spreadsheet programs is getting better, but there has never been anything that can fairly be called computer documentation, no manual that explains, to someone who doesn't already know, what a computer does. One can't document the inherently undefined.

Without that definition of purpose, sales are problematic. Apple sales skyrocketed when Personal Software released VisiCalc for the machine.

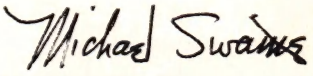
So one adopts a simplifying metaphor. The computer is a spreadsheet, the computer is a desktop, the computer is a vending machine: drop in a problem and out falls a solution. That's how technical writers document, that's how dealers sell, that's how users understand computers. One needs that grounding metaphor, and it's the software designer who provides it.

The Xerox PARC designers wove their homely office-furnishings metaphors and produced the Alto and Star, whence Apple's hopes for its corporate future in starspawned Macintosh and Lisa. Dan Bricklin and Bob Frankston spun the metaphor of the computer as spreadsheet and came up with VisiCalc. A computer plus VisiCalc is an electronic spreadsheet, something whose purpose can be understood and documented.

But every metaphor is a simplification, a fiction which, believed, closes doors. Metaphors can be hidden inside the words we use, and can subtly influence our thinking. Is COPY really a better operating system command name than PIP? While COPY seems to say clearly what it does, PIP is an acronym whose full name, Peripheral Interchange Program, reminds us that it is a program, written by an individual for a purpose not necessarily coextensive with our intuition about what copying is. Which metaphor serves us better?

The ideal is to create good metaphors and not to be controlled by unexamined metaphors. That's what distinguishes a software designer from a programmer. The software designer is one who can glimpse the computer, however briefly, as a computer, as a truly general-purpose device, and then create a new purpose, a new metaphor. The good programmer uses an appropriate metaphor effectively; the software designer steps out in the undefined air and weaves a metaphor to stand on.

We think that good, creative software design is important, and we want to promote it. Within the next two months we will be introducing a new department of the magazine, The Software Designer, in which working software designers will discuss basic issues in software design.



Michael Swaine

# Self-dFENSE™ for EDP managers.


The micro invasion has begun. And, chances are, you've now got a lot of different people in a lot of different departments using a lot of different micros.

Now there's a way for you to control and maximize the benefits of all the different micros in your domain.

## Fight back with dBASE II.®

dBASE II is the relational database management system from Ashton-Tate that enables you to manage your micro-based corporate data resources with the high level of consistency and sophistication you've enjoyed with mainframe and minicomputer systems.

Armed with dBASE II and the dBASE II RunTime™ program development module, you can write programs which will enable micro users in each department to "do their own thing" while creating complete database consistency throughout the company.

dBASE II is a powerful, flexible way for you to effectively manage the micro proliferation.

## Help is here.

If you'd like to know more about how dBASE II and RunTime can help you win the micro management battle, contact Ashton-Tate today. 10150 West Jefferson Boulevard, Culver City, CA 90230. (800) 437-4329, ext. 217 . In Colorado (303) 799-4900. In the U.K. (0908) 568866.

ASHTON·TATE™

## A Small-C Redundancy

Dear Dr. Dobb,

I'm a new subscriber to *DDJ* who has recently managed to obtain a copy of Jim Hendrix's Small-C compiler. When I got it up and running, I discovered that some functions produced extraneous, unreachable code—namely, functions that explicitly returned a value as the last statement in the function and that also declared local variables. For example,

```
function() {
   char c;
   return(c);
   }
```

The problem was that the explicit return caused stack-adjusting instructions to be generated along with the RET, to delete the local variable space. These same instructions were generated again at the end of the function (which in this case was a couple of characters later, immediately after the return).

I traced the source of the problem to the compound() function and designed a patch, which I sent to Jim Hendrix. During the course of our correspondence on this matter, he came up with a better patch, which is contained in the material following this letter. I've applied the patch, and it seems to work as indicated.

I'm grateful to Ron Cain, J.E. Hendrix, and *Dr. Dobb's Journal* for providing this compiler. I learned a lot from it, and, now that I have it working, intend to make good use of it.

Thanks,
David Bookbinder
888 Mass Ave. #511
Cambridge, MA 02139

*Mr. Hendrix writes:*

David Bookbinder has observed that Small C generates redundant code for deallocating local variables at the end of a function when the last statement is a return. The same is true if a return is the last statement in any compound statement which declares local variables. This is harmless, but it unnecessarily adds to the size of programs and it really is untidy.

Likewise, if the last statement in a function is a goto, unnecessary code for deallocating local variables and a RET is generated. Since a goto would block the exit path from a function, there is no need for such code. (The compiler already eliminates the RET if the last statement is a return.) There is no problem with goto's terminating nested compound statements since goto's are not allowed if variables are declared anywhere after the opening brace of the function body. So there are no variables to deallocate.

The following patch will eliminate these instances of redundant and unnecessary code.

(1) Add the following line to the end of the file CC.DEF:

```
#define STLABEL 14
```

(2) Change the line in newfunc() (file CC12.C) which contains a call to statement() to read:

```
statement():
#ifdef STGOTO
if(lastst != STRETURN &&
lastst != STGOTO) ffret();
#else
if(lastst != STRETURN) ffret();
#endif
```

(3) Change the line in statement() (file CC13.C) which contains a call to dolabel() to read:

```
else if(dolabel())
lastst=STLABEL;
```

(4) Change the line in compound() (file CC13.C) which contains a call to modstk() to read:

```
#ifdef STGOTO
   if(lastst != STRETURN &&
   lastst != STGOTO)
#else
   if(lastst != STRETURN)
#endif
   modstk(savcsp, NO);
   /* delete local variable space */
csp=savcsp;
```

(5) Recompile, assemble, and link the compiler.

## He Found Problems, Too

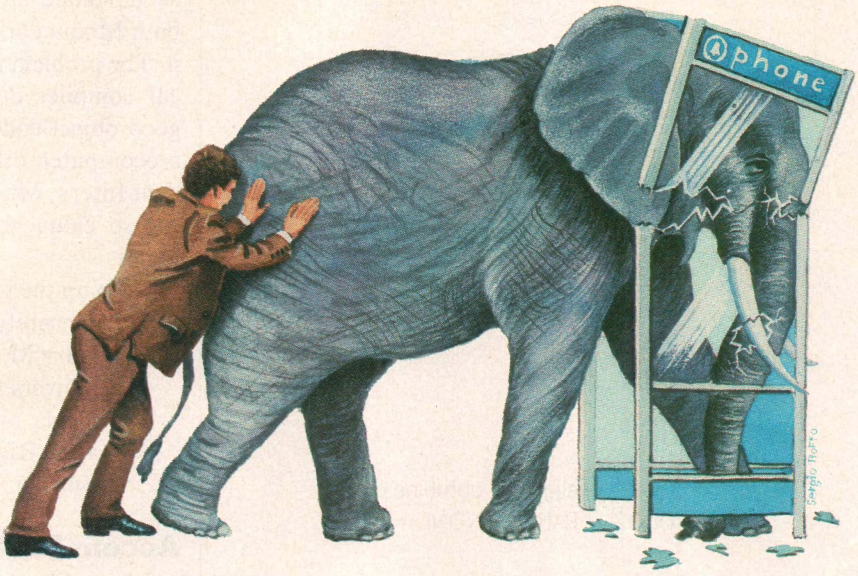
Dear Editor:

I read with considerable interest Ray Duncan's comments regarding the Microsoft Macro Assembler for the IBM PC in your February 1984 issue. Since the State University of New York at Buffalo is acquiring IBM PC's at rather a good clip, I decided to experiment a bit with the assembler under the theory that if there are that many bugs, there may well be many more. As it turned out, the first thing I tried failed. Basically, as the code in Listing One (page 12) demonstrates, the assembler seems to ignore the fact that IF-nesting is local to MACRO's (including IRP's, etc.) and *all* nested unsatisfied IFs must be cleared if the MACRO is exited via EXITM. If the seemingly "extra" ENDIF is removed (the line with the !'s), the assembler reports unclosed IF's on both passes. With the extra ENDIF, it assembles correctly, but this pattern of performance makes EXITM almost useless.

Further experiments with IRP's and EXITM's enclosed within MACRO's yielded results too bizarre to interpret simply or consistently. This situation, coupled with what Mr. Duncan observed, puts this piece of software firmly in the "Not Recommended" category insofar as SUNY at Buffalo is concerned!

I do disagree somewhat with Mr. Duncan's observation that a "more unlikely high level language" than Pascal could not have been chosen to write an

# Squeezing A Large Program Into A Small Memory Space?




It's time you got Plink86,™ the overlay linkage editor that's bringing modular programming to Intel 8086-based micros.

With Plink86,* you can write a program as large and complex as you want and not worry about whether it will fit within available memory constraints. You can divide your program into any number of tree-structured overlay areas. 4095 by 32 deep. Work on modules individually. Then link them into executable files. All without making changes to your source program modules.

Use the same module in different programs. Experiment with changes to the overlay structure of an existing program. Use one overlay to access code and data in other overlays.

Plink86 is a two-pass linkage editor that links modules created by the Microsoft assembler or any of Microsoft's 8086 compilers. Plink86 also works with other popular languages, like Lattice C, C86, or mbp/COBOL. And supports symbolic debugging with Phoenix' Pfix86 Plus.™

Plink86 includes its own object module library manager – Plib86™ – that lets you build libraries from scratch. Add or delete modules from existing libraries...Merge libraries... Or produce cross-reference listings.

Why squeeze any more than you have to? Plink86 by Phoenix. $395. Call (1) 800-344-7200. Or, write.

Phoenix

**Phoenix Software Associates Ltd.**
1420 Providence Highway Suite 260
Norwood, MA 02062
In Massachusetts (617) 769-7020

*Plink86 will run under PC DOS, MS-DOS™ or CP/M™-86.

## Listing One

### Microsoft Assembler Illustration

```
The IBM Personal Computer MACRO Assembler 02-8-84  PAGE  1-1


                                ASSUME CS: CSEG
0000                    CSEG    SEGMENT
                                IRP  R,<B,C,D>
                                IFIDN <R>,<C>
                                DB   1
                                EXITM
                                ELSE
                                DB   0
                                ENDIF
                                ENDM
0000  00                +       DB   0
0001  01                +       DB   1
                                ENDIF  ; !!!!
0002                    CSEG    ENDS
                                END

Segments and groups:

                Name            Size    align   combine class
CSEG . . . . . . . . . . . . .  0002    PARA    NONE

Warning Severe
Errors  Errors
0       0
```



assembler in. After all, Microsoft could have chosen COBOL! (Such a choice is not quite as impossible as it might seem; back in the mid-1960's the French wrote part of an Algol compiler for the IBM 7040/44 series in COBOL.)

Seriously, Pascal is not a *bad* choice as language structures go; after all, both Modula and Ada descended from it. The problem is that the IBM PC Pascal compiler does not generate very good object code. This is a global microcomputer difficulty, as your Resident Intern, Mr. Cortesi, has been stating so eloquently in the last several issues.

Keep up the good work!

Sincerely,
Gary M. Gibson, Assoc. Dir.
University Computing Services
SUNY at Buffalo
4250 Ridge Lea Road
Buffalo, NY 14226

## Accent Finder Fix

Dear Sirs,

It goes without saying that I was thrilled to see the "Accent Finder" in the May issue. In order to accommodate the changes you suggested, I had sent you three versions. The version published seems to be a combination of versions two and three. In its present shape it will work. However, it will fail to check for invalid characters within the word that's being analyzed. In order to remedy this, one need add only one line to the PROMPT procedure which should appear as shown in Listing Two (page 14).

For anyone interested in contacting our club, the UCLA IBM4341 Users' Group, we are on the BitNet. Our node is UCLAVM and my handle is VSS2364 (VSS2364 @ UCLAVM).

Long live Dr. Dobbs!

Yours truly,
Eddy C. Vasile
3314 Sawtelle 28
Los Angeles, CA 90066

## More RSA Remarks

Dear Editor:

I am responding to the letters that you published in the June 1984 issue regarding my recent article on the RSA Public Key System.

# Most Program Editors Are Shockingly Primitive.


Use Pmate™ once, and you'll never go back to an ordinary text editor again. Pmate is more than a powerful programmer's text processor. It's an interpretive language especially designed for customizing text processing and editing.

Just like other powerful editors, Pmate* features full-screen single-key editing, automatic disk buffering, ten auxiliary buffers, horizontal and vertical scrolling, plus a "garbage stack" buffer for retrieval of deleted strings. But, that's just for openers.

What really separates Pmate from the rest is macro magic. A built-in macro language with over 120 commands and single-keystroke "Instant Commands" to handle multiple command sequences. So powerful, you can "customize" keyboard and command structure to match your exact needs.

Get automatic comments on code. Delete comments. Check syntax. Translate code from one language to another. Set up menus. Help screens. You name it.

And, Pmate has its own set of variables, if-then statements, iterative loops, numeric calculations, a hex to decimal and decimal to hex mode, binary conversion, and a trace mode. You can even build your own application program right inside your text processor.

So, why work with primitive tools any longer than you have to? Pmate by Phoenix. $225. Call 1-800-344-7200. Or Write.

Phoenix

**Phoenix Software Associates Ltd.**
1420 Providence Highway Suite 260
Norwood, MA 02062
In Massachusetts (617) 769-7020

*Pmate is designed for microcomputers using the Intel 8086 family of processors, and running MS-DOS.™ A custom version is available for the IBM PC, TI Professional, Wang Professional, DEC Rainbow, and Z80 running under CP/M.™

## Listing Two

### Accent Finder Fix

```
procedure prompt(var word:WordType; var Error:boolean);
  begin
    write('Enter the word > ');
    readln(word);
    word:=UpperCase(Condense(word));
    CheckChars(Error,word)
  end;
```

First let me comment on remarks by Dr. Criscione. He points out that factoring of large prime numbers, in this case Mersenne primes, is being done. I pointed out in my article that a means of factoring large prime numbers would render the RSA system less useful. However, I believe that these primes are special cases and that the methods being used are specific to this set of primes. I should also point out that Knuth discusses methods of factoring primes in the book that I referenced in my article.

Next let me address remarks by Mr. Evenden. He states that RATFOR and FORTRAN are dinosaurs. Let me paraphrase Sam Clemens: The announcement of FORTRAN's death is premature. All languages have their pros and cons; FORTRAN, Pascal, C, etc., are no exceptions. I do not want to get into a debate over what language is best, but just want to say that languages are tools and any tool can be misused. I can just state that RATFOR/FORTRAN was what I had available at the time. Besides, FORTRAN is still the mainstay of the scientific and engineering community, good or bad.

He has submitted some C source that shows division/mod-less operations for the Multiple Precision Add/Subtract Algorithm implementation. I stated that all of the routines could certainly be improved. I thank him for his improvement and I welcome other suggestions.

He is correct that a typographic error exists in the Multiply Algorithm at step M5 (March 1984 *DDJ*, page 19). Instead of i=i–i it should read i=i–1. In addition, he is absolutely correct in a strict mathematical sense that INT(x) is not equivalent to FLOOR(x). However, as he states, it is true for non-negative integers (natural numbers). I pointed out that all of the mathematics of the RSA system would be limited to natural numbers, thus its equivalence in this case is valid.

Since I'm writing, I'll mention the few additional errors I have found. In Part I (March 1984), there is an underscore missing at the end of a line in Figure 1, page 17. The line (somewhat abbreviated) should read: "longline=variable1*(cos( . . . )+variable4* __". Also, in Figure 2b the "— or —" should have come *after* the line "= 16 mod 7 = 2", not before.

There are also three code alignment problems. In Part I, the nine lines of code at the top of page 36 (Listing Four) should be moved right one tab position so that the line "carry=0 . . ." aligns with the line "idxn2=len2 . . . ." Additionally, the top four lines on page 38 (Listing Four) should be moved right two tab positions so that the brace in line four of that page is indented from the "else" that follows it. In Part II (April 1984), the code block at the bottom third of page 57 (Listing Eight) which reads "do idx=1,8 # convert lower case to upper case" should be moved right one tab position to align with the line "read(CONSOLE,300) . . . ."

I hope this information helps.

Yours truly,
Charles E. Burton
1720 S. Deframe Ct.
Denver, CO 80228

**DDJ**

# Still Fixing Bugs The Hard Way?

Ready to take the sting out of debugging? You can with Pfix86™ and Pfix86 Plus™, the most advanced dynamic and symbolic debuggers on the market today for PC DOS and MS-DOS™ programmers.

What other debugger offers you an adjustable multiple-window display so you can view program code and data, breakpoint settings, current machine register and stack contents all at the same time? And, an in-line assembler so you can make program corrections directly in assembly language. Plus, powerful breakpoint features that allow you to run a program at full speed until a loop has been performed 100 times, or have the program automatically jump to a temporary patch area.

Or maybe you're tired of searching through endless piles of listings for errors? With Pfix86 Plus you won't have to. You can locate instruction and data by the symbolic name and using the symbolic address. Handle larger, overlayed programs with ease. And, Pfix86 Plus is designed to work with our Plink86™ linkage editor.

But that's not all. With a single keystroke you can trace an instruction and the action will be immediately reflected in code, data, stack, and register windows. Pressing a different key will elicit a special trace mode that executes call and loop instructions at full speed, as though only a single instruction were being executed.

And you get an easily accessible menu that makes the power of our debuggers instantly available to the new user, but won't inhibit the practiced user.

So, why struggle with bugs? Pfix86 by Phoenix. Pfix86 $195. Pfix86 Plus $395. Call (1) 800-344-7200, or write.

*Phoenix*

**Phoenix Software Associates Ltd.**
1420 Providence Highway Suite 260
Norwood, MA 02062
In Massachusetts (617) 769-7020

# DR. DOBB'S CLINIC

by D.E. Cortesi

## The Missing RAM-drive

In April we passed along Malcom Fordham's request for help. He uses CP/M-86 on an IBM PC and wanted to set aside some of the PC's storage for a RAM drive, but he couldn't find the software to do it.

Well, it turns out that the answer was right there in his own system. U. Thier of Bloomfield, CT, wrote to point out that CP/M-86 for the PC, as shipped by Digital Research, contains support for a RAM drive. We and Fordham had overlooked it because setting up the RAM drive is a function of the SETUP command. That command is documented only in the "Release Note," an unimposing little pamphlet at the back of the manual binder.

The Release Note looks like some kind of techie's afterthought, and quite a few users of CP/M-86 probably have never looked into it. That's unfortunate, because only there does one learn how to do such useful things as:

- Setting a command line that will be executed at power-up (you could set the line "submit profile," for example)
- Saving the configurations of the serial ports and function keys so you don't have to reset them after a cold boot
- Changing the disk step-rate, thus speeding up the system quite a bit
- Installing a special device driver

And, of course, setting the base address and size of a segment of storage to be used as drive M:. If you use CP/M-86 on the PC, take a close look at that Release Note and the SETUP command it describes.

## What Day Is This?

This column started out three years ago (good grief, is it that long?) with a discussion of calendar algorithms. Apparently the message didn't get through to somebody (sigh, nobody listens to us), because Glenn Roberts of Knoxville, TN, has written to tell us of a calendar error in Lotus 1-2-3.

"I am always surprised how many sophisticated programmers do not really know the rules of the calendar," Roberts says. "Webster's New World Dictionary defines the Gregorian Calendar as follows:

a corrected form of the Julian calendar, introduced by Pope Gregory XIII in 1582 and now used in most countries of the world: it provides for an ordinary year of 365 days and a leap year of 366 days every fourth even year, exclusive of century years, which are leap years *only if exactly divisible by 400.* [emphasis added]

Thus the year 1900 was not a leap year (even though it is divisible by 4), but the year 2000 will be.

"The lack of understanding of this simple set of rules has shown up as bugs in some programs. Lotus 1-2-3, for example, contains a built-in function @DATE, which handles the years 1900 and 2000 incorrectly. For 1900 it thinks there *is* a February 29th, while for 2000 it thinks there *isn't* one—just the opposite of the truth. The internal representation of the date maintained by Lotus is supposed to be the number of days since December 31, 1899; however, it is actually one more than this for dates between 3/1/1900 and 2/29/2000."

Roberts goes on to say that it would be interesting to hear of other popular programs with calendar bugs. We think so, too.

## @In-, Out-, and Throughput

R. C. Wagner, of Indecipherable, CA, wants to know what exactly we mean by "a throughput of $x$ bytes per second," a phrase we used last November. It's a good question. We've never seen a formal definition but rather intuited the meaning of the word "throughput" from the contexts it appears in. Now that Wagner made us think about it, the idea has some holes.

Given a program whose input is from, and output is to, secondary storage (tape or disk) and given the need to say something meaningful about its total performance, you have two choices. You can talk about its elapsed runtime, but you always have to qualify that with the quantity of input (or output) data it processed in that run. You have to say something like: "It took 34 seconds to process a 73-kilobyte input file."

It's clearer to talk instead about the program's "throughput," meaning nothing more than the quantity of data it processed divided by the time it took to process it. That yields a figure of kilobytes per second. It's a simple measure and one that allows another person to project how long the program would take to process a file of some other size.

Well, maybe. When you really think about the idea, you realize that it rests on a lot of unstated assumptions that might not be true. Let's examine some of them.

First, throughput includes input time, output time, and all the processing in between. Since input, output, and processing all contribute to the total runtime, throughput measures the program's gross behavior. But it doesn't say anything about which of the three parts dominates. If the program is too slow (do we ever worry about a program because it's too fast?), knowing its throughput gives us no clue as to where to start looking for a bottleneck.

Second, if the program does little processing, its input and output speed will dominate its throughput rate. I/O speed is often determined by factors external to the program—the operating system, the device drivers, and the speed of the devices themselves. When

a throughput rate is quoted, the unstated assumption is that these external factors are constant. Of course, they are only constant within a single system not between systems (as we will see very shortly).

Third, there is an unstated assumption that the program runs in time that is a linear function of the quantity of data it processes; that is, if the amount of data is exactly doubled, the runtime will also exactly double. That's probably true of the I/O portions of the runtime but might be completely false as a description of the processing time. Programs exist whose processing time is *exponential* in the quantity of data they process. It would be quite deceptive to quote a throughput figure for such a program, since the "throughput rate" would change drastically with the amount of data—just the opposite of the accepted meaning.

Sort routines never have a linear processing time function, but that may not matter if sorting is only a small part of what a program does or if the sort time is very small relative to the I/O time. But many programs display nonlinear profiles when runtime is graphed against data quantity. Compilers, for example, may begin to spill their internal tables to disk at a certain size of source program, with the result that their runtime graph has a distinct "knee" at that size of input.

In short, "throughput" is a valid, if rough, measure of performance for programs that produce output that is proportional in quantity to their input and whose processing time is either insignificant or known to be a linear function of the amount of data processed. (That pretty well describes a typical utility program.) Even when these conditions are met, the problem of comparing the I/O environment remains.

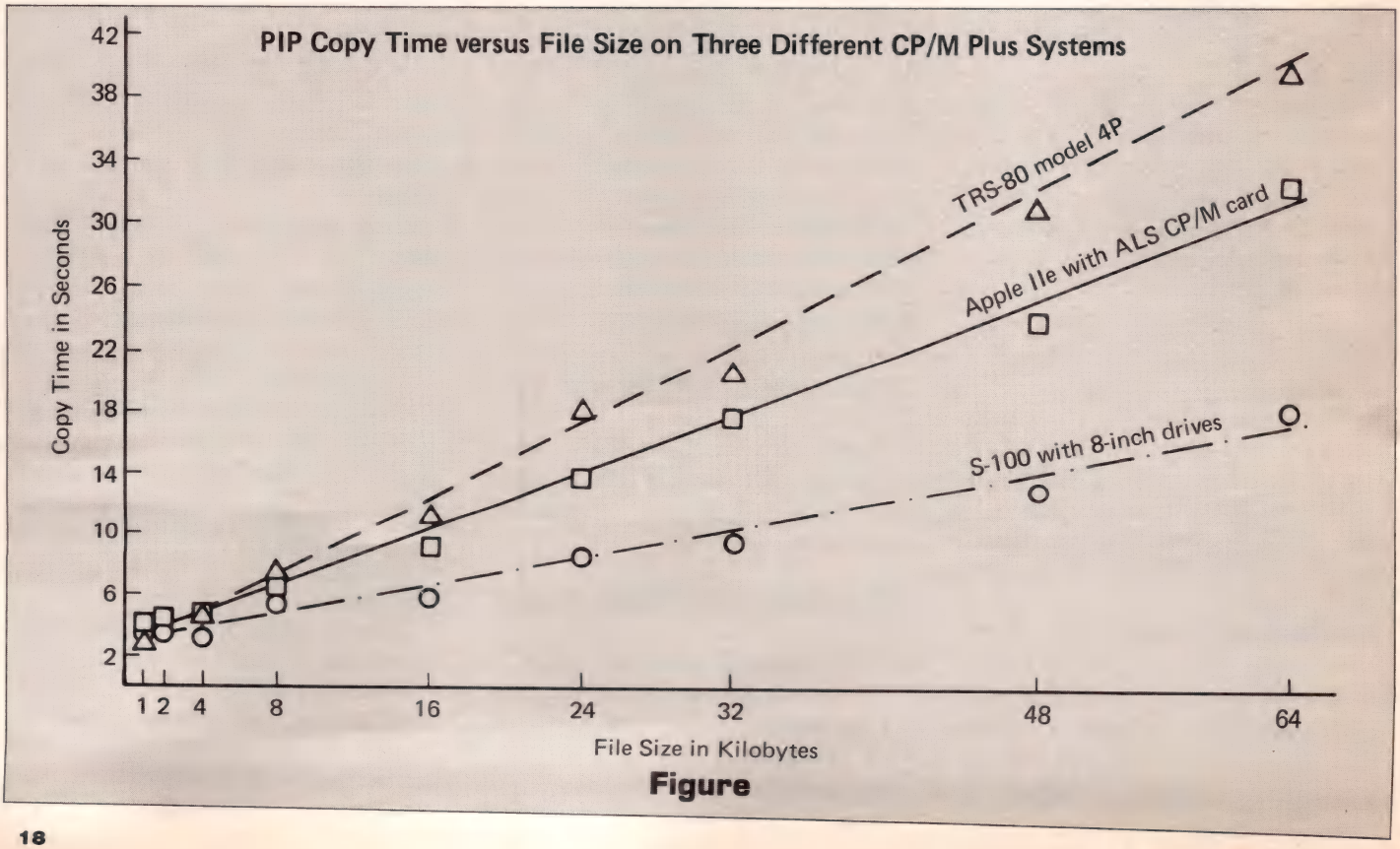
## @Into the Lab

After writing that insightful commentary, we decided to see if we really knew what we were talking about. By good fortune, we have access to three systems that run identical software but have very different hardware. One is an S-100 system with 8-inch disk drives; one is a TRS-80 model 4P; and the third is an Apple IIe with Advanced Logic Systems' CP/M card. All three run CP/M Plus and have Z80 CPUs, but the Apple's CPU runs at 6 MHz while the others run at 4 MHz.

On each machine we set up a series of ASCII files of different sizes. Then we copied the files with PIP and measured the time it took to copy each file on each machine. The copies were from one drive to the other. Program load time was not measured; in each case PIP was called with no arguments, and only the copy time was measured. After each measurement the output file was erased, so every output file started at the same place on the output disk. However, the input files were built from smallest to largest, so they started at different tracks (the largest starting farthest from the directory).

The figure (below) summarizes the results. As you can see, each machine's times plot out as a good approximation to a straight line. (That verifies that hypothesis that PIP's processing time is a linear function of the amount of data it handles — no great surprise.) The slope of a line is the throughput rate for a PIP copy in that hardware environment.

The table (page 19) lists the measured times, along with the throughput rate that would be calculated from each measurement in isolation. Examine these numbers and you'll discover some pitfalls to measuring throughput. The calculated throughput for small files bears little resemblance to that for larger files. In fact, the linear functions




**Figure**

that show up so clearly in the graph aren't obvious in the numbers of the table at all. The best-fit straight line for each machine seems to approximate most closely to the calculated throughput for a 32-kilobyte file.

There's a lesson here. We've said before that computer science is one of the few that allows precise, repeatable measurements, and there really is no excuse for making relative judgments without having the measurements to back them up. We were guilty of exactly that last November, when we cited a "throughput" value based on exactly one measurement—tsk, tsk.

There's also an idea lurking in the figure. Disk I/O time dominates so many things we do with personal computers, it would be nice to have some kind of disk I/O figure of merit with which we could compare one system to another. The throughput rate of a simple file copy looks as if it might work for that purpose.

Where would your machine's line fall in the graph of the figure? Well, why don't you take the measurements and find out? Remember, program load time isn't included, just the time from entering the file transfer instruction until the next prompt. On a CP/M system, that's from pressing enter on "*b:=a:filename" through the next PIP asterisk prompt. On an MS-DOS system, it would be from pressing enter on "copy filename b:" until the next system prompt (since the MS-DOS copy command is built in).

If enough people send in measurements like those in the table, we'll publish an amalgamated graph. As a way of comparing day-to-day useful performance, it would be a bit more meaningful than the Sieve of Eratosthenes.

By the way, R. C. Wagner's question, which started all this, was written in the comment space on a Reader Service card. That's great! Any communications medium that gets more input for this column is okay by the Intern—although if you enter the First & Last Annual ZOSO Sound-Alike contest that way, you're going to have to write awfully small. ■■J

| File size | S-100 system | | TRS-80 Model 4P | | Apple IIe/ALS | |
|---|---|---|---|---|---|---|
| (kilobytes) | time | thruput | time | thruput | time | thruput |
| 1 | 3.4 | 0.29 | 3.2 | 0.31 | 4.0 | 0.25 |
| 2 | 3.4 | 0.59 | 4.0 | 0.50 | 4.2 | 0.48 |
| 4 | 3.4 | 1.18 | 5.0 | 0.80 | 5.0 | 0.80 |
| 8 | 4.8 | 1.67 | 7.2 | 1.11 | 6.4 | 1.25 |
| 16 | 5.8 | 2.76 | 11.0 | 1.45 | 9.2 | 1.74 |
| 24 | 8.2 | 2.93 | 17.4 | 1.38 | 13.8 | 1.74 |
| 32 | 9.6 | 3.33 | 20.6 | 1.55 | 17.4 | 1.83 |
| 48 | 13.2 | 3.63 | 30.0 | 1.60 | 23.6 | 2.03 |
| 64 | 18.0 | 3.56 | 39.5 | 1.62 | 32.5 | 1.97 |

**Table.**
**The measured PIP copy times, from which the figure was drawn, and the calculated throughput rates that result**

**Reader Ballot**
Vote for your favorite feature/article.
Circle Reader Service **No. 190.**

# CP/M EXCHANGE

by Robert Blum

Searching for the right tools to fill your program development tool chest can be an exhausting experience; not to mention the damage to your wallet if you make any wrong decisions along the way. Fortunately, when looking for the right assembly language development package you will not have to contend with nearly as many competing packages as in other application areas. Nonetheless, deciding which package best suits your needs may be more difficult than necessary unless you first spend some time defining what your needs are and what those needs equate to in terms of dollars.

Before you go to the local software store to get a demonstration of the packages that they carry I suggest you first contact several manufacturers of packages that look interesting to better understand what you should expect to get for your money. I make this suggestion because I think you will find that very few software dealers know anything about the type of product you are looking for. And in most cases, even if you are fortunate enough to find a dealer that actually has this type of software available for purchase, you will probably have to read the manual to figure out how to run your own demonstration.

### Getting Started

Most assembly language development packages fall into one of two categories. The under $50.00 package is aimed at the hobbyist or the infrequent assembly programmer. These packages are usually limited to an assembler that will accept Intel 8080 mnemonics and possibly extensions for the Z80. Occasionally one can be found that also offers the Zilog set of mnemonics for the Z80. On the opposite end of the spectrum is the commercial package priced at upwards of $150.00 or more. Packages of this type usually offer a large number of convenience features that mainly appeal to the system developer or systems house. Included in a package of this type you should expect to find at least a macro assembler that recognizes several mnemonic dialects and a linkage editor to be used in combining separately assembled subroutines. You may also find some useful utilities such as a cross-reference program.

My own experience in finding the right assembly language development package has ranged from downright horror to moderate satisfaction. Although I must qualify that statement by saying that by the time microcomputers first appeared I had already been spoiled by over 10 years of experience with IBM's mainframe macro assembler. Since that time my satisfaction with it has not wavered and I continue to use it as my basis for comparison. Don't let the word "mainframe" unduly influence your thoughts; I am not about to begin comparing apples to oranges.

### In the Beginning

To put things into the proper perspective: I was introduced to IBM's macro assembler on a 32K 360/30. For those of you who have a mainframe background, thoughts of this grand old machine will undoubtedly bring back many fond memories. The standard macro assembler included at no charge with this machine required only a 14K partition of memory to run. If more memory was available it would be used, but only this minimal amount was necessary. If the machine you were using was blessed with a full 64K of memory, a 44K variant of the 14K assembler could be used. Both assemblers offered identical features, only the 44K version executed three or four times faster because fewer overlays were used and more memory was available for table space.

Through the years this same assembler has been transported from generation to generation of IBM computer. With the exception of the additional instructions added with each new generation of processor, the assembler as I knew it has been retained. Certainly this longevity speaks highly of the original design and its acceptance by the user community.

Don't be surprised if the way I have described the 360/30 sounds similar to your microcomputer. The 360/30 CPU was almost as powerful as a 4MHz Z80, and the only area in which the mainframe surpassed today's microcomputer was in the power of its I/O system.

I have used a number of different assembly language development packages marketed by both large and small companies. Many of them offered an excellent value for the price, and a few offered some very exciting, unique features. However, none of them have even come close to satisfying my desire for the powerful and flexible macro facility I had become accustomed to. Practically all of the macro assemblers I have looked at use the Microsoft macro format as a starting point. Any additions are then made around that standard. I have nothing against the Microsoft macro format, as far as it was taken. I simply think that it was considered complete long before it had matured.

Nonetheless, I was unable to find anything better than M80 and out of desperation chose it for my assembler. Actually, my choice was not quite as difficult as I make it sound. M80 is provided as part of a utilities package that is included at no extra charge when you purchase other Microsoft language products. Since I couldn't find anything better than M80 and obviously couldn't beat the price, my decision was easy.

### Something New

At CP/M 83 I had the pleasure of meeting Steve Russell, the S and R of SLR Systems. Steve was proudly ex-

hibiting his two latest creations, Z80ASM and Z80LNK. Both are included in an assembly language development package that was purported to be six times faster than M80. I am not a foreigner to M80, and I have never been willing to accept at face value what appears to be an outrageous advertisement, so I asked Steve to send me a review package. After some conversation he asked me if I would have any interest in serving as a Beta test site for Z80ASM after he had added a few more features to make it compatible with M80. I couldn't possibly turn down an offer like that and I left the show questioning Steve's conviction that Z80ASM actually executes six times faster than M80. Don't get me wrong; I'm not suggesting that M80 is fast, just that a speed improvement of two or three times would have been more believable.

Around the first of the year the UPS truck stopped to drop off an unexpected package. To my surprise it was from SLR Systems. I had totally forgotten about Z80ASM and my conversation with Steve. Opening the package revealed a three-ring binder of documentation and two disks. The first disk contained the assembler, linkage editor, and installation programs. The other disk contained a number of sample source programs to be used in testing the assembler.

### Egg on the Face

While furiously thumbing through the documentation to find the instructions for installing the assembler, I began to chuckle about how easy I thought it was going to be to disprove the advertising claims made about Z80ASM. A few minutes later I had finished configuring the assembler for my system and was ready to run my first test. Rather than using one of the sample source programs included with Z80ASM I chose to use one of my largest programs.

In my haste to get started I had not taken time to thoroughly read the documentation; I ended up blindly specifying every runtime option that looked meaningful. With my stopwatch in hand and a smirk on my face I pressed the return key and anxiously waited for something to happen. And happen it did! After only a few seconds a message was displayed on the CRT suggesting that the first pass was now complete and that the second pass through my program was now in progress. My first thought was that something must have gone wrong because I couldn't believe that the assembler had completed the first pass this quickly. I began to feel a little better when the assembler appeared to be running the second pass much more slowly.

After 25 seconds the end-of-job statistics were displayed on my CRT and control was returned to CP/M. After making a note of the elapsed time taken by Z80ASM I ran the same source program through M80. At the conclusion of the M80 assembly I was satisfied with my suspicion that the claims made about Z80ASM were indeed overstated because M80 took only a little over twice as long as Z80ASM. What I didn't realize was that specifying all the runtime options for Z80ASM commands the program to also create an output listing file including a cross-reference and symbol table. Because I had used one of my largest programs for the test, a listing file of over 160K had been created in addition to the normal .SYM and .REL files. I was suddenly a little more seri-

ous about running a complete benchmark after realizing that all of this processing had taken place in 25 seconds. To complete the test with fairness I ran the cross-reference program provided with M80 and directed its output to disk. I then added the runtime of the M80 assembly to that of the cross-reference program to get an overall total.

When I compared the two runtime totals I discovered that my suspicions were ill-founded and that SLR Systems had not overstated the performance of its product. Straight out of the box I achieved the claimed sixfold runtime reduction.

Next month I will talk more about how Z80ASM does its job.

### Expert Marriage Counseling

I won't be telling you anything new when I point out that some incompatibilities exist between CP/M 2.2 and its successor, CP/M Plus. Dave Cortesi and I have commented on this in past columns, but both of us failed to provide any solution to the problem. Fortunately, Mike Griswold of Fort Worth, Texas, took the bull by the horns and wrote a Resident System Extension (RSX) that translates version 2.2 BIOS calls to those compatible with CP/M Plus.

After keying it in and correcting the errors that I introduced, I attached it to the public domain DU program. I chose this program to test with because I am familiar with its operation and the reason it fails when run under CP/M Plus. As Mike reported, DU now works perfectly, as do several other programs that I had shelved because of compatibility problems.

The full text of Mike's program can be found in the listing on page 23. It is a great piece of code that is an absolute necessity in warding off many of the mysterious conversion pains likely to occur when going from version 2.2 to CP/M Plus.

*Editor's note: Interested readers may also wish to consult the article on using RSXs under CP/M Plus, elsewhere in this issue.* **DDJ**

**Reader Ballot**
Vote for your favorite feature/article.
Circle Reader Service **No. 191.**

# CP/M Exchange Listing (Text begins on page 20)

```
;
        title   'CP/M 2.2 BIOS RSX'
;
;       18Jan84         By Mike Griswold
;
;       This RSX will provide CP/M 2.2 compatible BIOS support
;       for CP/M 3.x.  Primarily it performs logical sector
;       blocking and deblocking needed for some programs.
;       All actual I/O is done by the CP/M 3.0 BIOS.
;
        maclib  z80             ; Z80 opcode equates
        cseg
;
;       This equate is the only hardware dependent value.
;       It should be set to the largest sector size that
;       will be used.
;
max$sector$size:        equ     1024
;
;
;       RSX prefix structure
;
        db      0,0,0,0,0,0
entry:  jmp     boot
next:   db      jmp             ; jump
        dw      0               ; next module in line
prev:   dw      0               ; previous module
remove: db      0ffh            ; remove flag
nonbnk: db      0
        db      'BIOS2.21'
        db      0,0,0
;
;       Align jump table on next page boundary.  This is needed
;       for programs that cheat when getting the addresses of
;       BIOS jump table entries.  Optimization freaks could move
;       some code up here.  With a 60K TPA though its hard to
;       get excited.
;
        ds      229
;
;       BIOS Jump Table
;
cbt:    jmp     wboot           ; cold boot entry
wbt:    jmp     wboot           ; warm boot entry
        jmp     xconst          ; console status
        jmp     xconin          ; console input
        jmp     xconout         ; console output
        jmp     xlist           ; list output
        jmp     xauxout         ; aux device output
        jmp     xauxin          ; aux device input
        jmp     home            ; home disk head
        jmp     seldsk          ; select drive
        jmp     settrk          ; select track
        jmp     setsec          ; select sector
        jmp     setdma          ; set dma address
        jmp     read            ; read a sector
```

*(Continued on page 26)*


DO YOU
HAVE A
BYTE KIND
OF MIND?
THEN WE'VE
GOT YOUR KIND
OF COMPUTER
SHOW.
DON'T MISS
THE BYTE
COMPUTER
SHOW.
SEPT. 6-9
BROOKS
HALL AND
CIVIC AUDITORIUM.

# THE BYTE COMPUTER SHOW/San Francisco '84
## a unique and informative 25-session conference.

### SCHEDULE BY GROUP

| KEYNOTE | DAY | TIME |
|---|---|---|
| KN-1 Bit-Pusher Perspectives | 9/6 | 11:00-12:30 |
| **HARDWARE HELPERS** | | |
| HH-1 Who Needs 32 Bits? | 9/7 | 11:00-12:30 |
| HH-2 Is PC Compatibility Holding Us Back? | 9/7 | 5:00- 6:30 |
| HH-3 Adding-On For A Supercharged System | 9/8 | 2:00- 3:30 |
| HH-4 The 1200 bps Modem: Users Report | 9/8 | 5:00- 6:30 |
| **SOFTWARE SAVINGS** | | |
| SS-1 User Agreements: A New Day Dawning? | 9/6 | 2:00- 3:30 |
| SS-2 Programming Environments: New Tools and Techniques | 9/7 | 5:00- 6:30 |
| SS-3 The Home-Brew Data Base: Tips for Home Brewers | 9/8 | 5:00- 6:30 |
| **LANGUAGE LABORATORY** | | |
| LL-1 Micro Language Forum | 9/6 | 5:00- 6:30 |
| LL-2 C Language Tradeoffs | 9/7 | 2:00- 3:30 |
| LL-3 BASIC: Can It be Saved? | 9/8 | 11:00-12:30 |

| APPLICATIONS FRONTIER | DAY | TIME |
|---|---|---|
| AF-1 Home/Family Management: Beyond the Recipe Collection | 9/6 | 2:00- 3:30 |
| AF-2 Your Personal Robot | 9/7 | 11:00-12:30 |
| AF-3 Systems for the Handicapped | 9/7 | 2:00- 3:30 |
| AF-4 When Less is More: Notebook Computers | 9/8 | 11:00-12:30 |
| **SOFTWARE HORIZONS** | | |
| SH-1 Next Generation OS: Are Icons Inevitable? | 9/6 | 5:00- 6:30 |
| SH-2 Beyond Words: Idea Processing | 9/7 | 11:00-12:30 |
| SH-3 AI Gateways to Natural Languages | 9/7 | 2:00- 3:30 |
| SH-4 Voice Pattern Recognition | 9/8 | 2:00- 3:30 |
| **GRAPHICS GALORE** | | |
| GG-1 Keyboard Alternatives | 9/6 | 5:00- 6:30 |
| GG-2 Low Bucks Graphics Add-Ons | 9/7 | 5:00- 6:30 |
| GG-3 Micro Graphics Applications | 9/8 | 11:00-12:30 |
| **THE BEST IS YET TO COME** | | |
| YC-1 Coming Attractions: The Computer/Video Interface | 9/6 | 2:00- 3:30 |
| YC-2 Japanese Computer Trends | 9/8 | 2:00- 3:30 |
| YC-3 Mass Storage Alternatives | 9/8 | 5:00- 6:30 |

# CP/M Exchange Listing (Listing Continued, text begins on page 20)

```
        jmp     write           ; write a sector
        jmp     xliststs        ; list status
        jmp     sectran         ; sector translation
;
;       The CP/M 3.0 BIOS jump table is copied here
;       to allow easy access to its routines.  The disk
;       I/O routines are potentially in banked memory
;       so they cannot be called directly.
;
xwboot: jmp     0               ; warm boot
xconst: jmp     0
xconin: jmp     0
xconout:jmp     0
xlist:  jmp     0
xauxout:jmp     0
xauxin: jmp     0
        jmp     0
        jmp     0
        jmp     0
        jmp     0
        jmp     0
        jmp     0
        jmp     0
xliststs:jmp    0
;
;       Signon message
;
signon: db      0dh,0ah,'BIOS ver 2.21 ACTIVE',0dh,0ah,0
;
;       Cold boot
;
boot:   push    psw             ; a BDOS call is in progress
        push    h               ; so save CPU state
        push    d
        push    b
        lxi     h,next          ; now bypass this RSX on
        shld    entry+1         ; all subsequent BDOS calls
        call    init            ; initialize BIOS variables
        lhld    1               ; save the CP/M 3.0 BIOS jump
        shld    old$addr        ; at location 0
        lxi     d,xwboot        ; set up to move jump table
        lxi     b,15*3          ; byte count
        ldir
        lxi     h,wbt           ; substitute new jump address
        shld    1
        lxi     h,signon        ; sound off
        call    prmsg
        pop     b               ; restore BDOS call state
        pop     d
        pop     h
        pop     psw
        jmp     next            ; carry on
;
;       Warm boot
;
wboot:  lhld    old$addr
```

```
        shld    1               ; restore normal BIOS address
        jmp     0               ; jump to CP/M 3.0 warm boot
;
;       Initialize BIOS internal variables for cold boot
;
init:   xra     a
        sta     hstwrt          ; host buffer written
        sta     hstact          ; host buffer inactive
        lxi     h,80h
        shld    dmaadr
        ret
;
;       Routine to call banked BIOS routines via BDOS
;       function 50.  All disk I/O calls are made through
;       here.
;
xbios:  sta     biospb          ; set BIOS function
        mvi     c,50            ; direct BIOS call function
        lxi     d,biospb        ; BIOS parameter block
        jmp     next            ; jump to BDOS
;
biospb: db      0               ; BIOS function
areg:   db      0               ; A register
bcreg:  dw      0               ; BC register
dereg:  dw      0               ; DE register
hlreg:  dw      0               ; HL register
;
;       Home disk.
;
home:   lda     hstwrt          ; check if pending write
        ora     a
        cnz     writehst        ; dump buffer to disk
        xra     a
        sta     hstwrt          ; buffer written
        sta     hstact          ; buffer inactive
        sta     unacnt          ; zero alloc count
        sta     sektrk          ; zero track count
        sta     sektrk+1
        ret
;
;       Select disk.  Create a fake DPH for programs
;       that might use it.
;
seldsk: mov     a,c             ; requested drive number
        sta     sekdsk
        sta     bcreg           ; set C reg in BIOSPB
        mvi     a,9             ; BIOS function number
        call    xbios           ; CP/M 3.0 select
        mov     a,h
        ora     l               ; check for HL=0
        rz                      ; select error
        mov     e,m             ; get address of xlat table
        inx     h
        mov     d,m
        xchg
        shld    xlat            ; save xlat address
        lxi     h,11            ; offset to dpb address
        dad     d
        mov     e,m             ; fetch address of dpb
```

*(Continued on next page)*

```
        inx     h
        mov     d,m
        xchg
        shld    dpb             ; address of dpb
        mov     a,m             ; cpm sectors per track
        sta     spt
        inx     h
        inx     h               ; point to block shift mask
        inx     h
        mov     a,m
        sta     bsm             ; save block shift mask
        lxi     d,12            ; offset to psh
        dad     d
        mov     a,m
        sta     psh             ; save physical shift factor
        lxi     h,dph           ; return DPH address
        ret
;
;       This fake DPH holds the addresses of the actual
;       DPB.  The CP/M 3.0 DPH is *not* understood
;       by CP/M 2.2 programs.
;
dph:    equ     $
        dw      0               ; no translation
        ds      6               ; scratch words
        ds      2               ; directory buffer
dpb:    ds      2               ; DPB
        ds      2               ; CSV
        ds      2               ; ALV
;
;       Set track.
;
settrk: sbcd    sektrk
        ret
;
;       Set dma.
;
setdma: sbcd    dmaadr
        ret
;
;       Translate sectors.  Sectors are not translated yet.
;       Wait until we know the physical sector number.
;       This works fine as long as the program trusts
;       the BIOS to do the translation.  Some programs
;       access the XLAT table directly to do their own
;       translation.  These programs will get the wrong
;       idea about the disk skew but it should cause no
;       harm.
;
sectran:mov     l,c             ; return sector in HL
        mov     h,b
        ret
;
;       Set sector number.
;
setsec: mov     a,c
```

*(Continued on page 30)*

# PRESENTING
# The first compiler for dBASE II®


WordTech Systems is proud to announce the first compiler for dBASE II®. And we are introducing it with a special offer.

## INDEPENDENCE

Now you can write compiled, efficient programs that will execute independently of dBASE II, and without RunTime®.

## NO LICENSE FEES

You only buy dB Compiler™ once. You may compile as many applications as you wish, FOREVER, with no additional fees.

## SPEED

Application programs are compiled into low level code and only include program functions that are absolutely necessary.

## SECURITY

Compilation is far better than encryption for protecting your programming insights and procedures.

## PORTABILITY

Using dB Compiler's cross-linkers you can use one development system to generate code for various target environments.

Suggested retail price: $750; additional target modules: $350
Special Offer: Compiler and an additional target module: $750
Offer expires 7/15/84. Corp/multi-user licenses available.

**dB COMPILER™**

WORDTECH SYSTEMS P.O. Box 1747, Orinda, CA 94563 (415) 254-0900

dBASE II, RunTime® Ashton-Tate

Circle **no. 76** on reader service card.

# CP/M Exchange Listing (Listing Continued, text begins on page 20)

```
        sta     seksec
        ret
;
;       Read the selected CP/M sector.
;
read:   mvi     a,1
        sta     readop          ; read operation
        inr     a               ; a=2 (wrual)
        sta     wrtype          ; treat as unalloc
        jmp     alloc           ; perform read
;
;       Write the selected CP/M sector.
;
write:  xra     a
        sta     readop          ; not a read operation
        mov     a,c
        sta     wrtype          ; save write type
        cpi     2               ; unalloc block?
        jrnz    chkuna
;
;       Write to first sector of unallocated block.
;
        lda     bsm             ; get block shift mask
        inr     a               ; adjust value
        sta     unacnt          ; unalloc record count
        lda     sekdsk          ; set up values for
        sta     unadsk          ; writing to an unallocated
        lda     sektrk          ; block
        sta     unatrk
        lda     seksec
        sta     unasec
;
chkuna: lda     unacnt          ; any unalloc sectors
        ora     a               ; in this block?
        jrz     alloc           ; skip if not
        dcr     a               ; unacnt=unacnt-1
        sta     unacnt
        lda     sekdsk
        lxi     h,unadsk
        cmp     m               ; sekdsk = unadsk ?
        jrnz    alloc           ; skip if not
        lda     sektrk
        lxi     h,unatrk
        cmp     m               ; sektrk = unatrk ?
        jrnz    alloc           ; skip if not
        lda     seksec
        lxi     h,unasec
        cmp     m               ; seksec = unasec ?
        jrnz    alloc           ; skip if not
        inr     m               ; move to next sector
        mov     a,m
        lxi     h,spt           ; addr of spt
        cmp     m               ; sector > spt ?
        jrc     noovf           ; skip if no overflow
        lhld    unatrk
        inx     h
```

```
        shld    unatrk          ; bump track
        xra     a
        sta     unasec          ; reset sector count
noovf:  xra     a
        sta     rsflag          ; don't pre-read
        jr      rwoper          ; perform write
;
alloc:  xra     a               ; requires pre-read
        sta     unacnt
        inr     a
        sta     rsflag          ; force pre-read
;
rwoper: xra     a
        sta     erflag          ; no errors yet
        lda     psh             ; get physical shift factor
        ora     a               ; set flags
        mov     b,a
        lda     seksec          ; logical sector
        lxi     h,hstbuf        ; addr of buffer
        lxi     d,128
        jrz     noblk           ; no blocking
        xchg                    ; shuffle registers
shift:  xchg
        rrc
        jrnc    sh1
        dad     d               ; bump buffer address
sh1:    xchg
        dad     h               ; double offset
        ani     07fh            ; zero high bit
        djnz    shift
        xchg                    ; HL=buffer addr
noblk:  sta     sekhst
        shld    sekbuf
        lxi     h,hstact        ; buffer active flag
        mov     a,m
        mvi     m,1             ; set buffer active
        ora     a               ; was it already?
        jrz     filhst          ; fill buffer if not
        lda     sekdsk
        lxi     h,hstdsk        ; same disk ?
        cmp     m
        jrnz    nomatch
        lda     sektrk
        lxi     h,hsttrk        ; same track ?
        cmp     m
        jrnz    nomatch
        lda     sekhst          ; same buffer ?
        lxi     h,hstsec
        cmp     m
        jrz     match
;
nomatch:
        lda     hstwrt          ; buffer changed?
        ora     a
        cnz     writehst        ; clear buffer

filhst: lda     sekdsk
        sta     hstdsk
        lhld    sektrk
```

*(Continued on next page)*

## CP/M Exchange Listing (Listing Continued, text begins on page 20)

```
        shld    hsttrk
        lda     sekhst
        sta     hstsec
        lda     rsflag          ; need to read ?
        ora     a
        cnz     readhst         ; yes
        xra     a
        sta     hstwrt          ; no pending write
;
match:  lhld    dmaadr
        xchg
        lhld    sekbuf
        lda     readop          ; which way to move ?
        ora     a
        jrnz    rwmove          ; skip if read
        mvi     a,1
        sta     hstwrt          ; mark buffer changed
        xchg                    ; hl=dma  de=buffer
;
rwmove: lxi     b,128           ; byte count
        ldir                    ; block move
        lda     wrtype          ; write type
```

```
        cpi     1               ; to directory ?
        jrnz    exit            ; done
        lda     erflag          ; check for errors
        ora     a
        jrnz    exit            ; don't write dir if so
        xra     a
        sta     hstwrt          ; show buffer written
        call    writehst        ; write buffer
exit:   lda     erflag
        ret
;
;       Disk read.  Call CP/M 3.0 BIOS to fill the buffer
;       with one physical sector.
;
readhst:
        call    rw$init         ; init CP/M 3.0 BIOS
        mvi     a,13            ; read function number
        call    xbios           ; read sector
        sta     erflag
        ret
;
;       Disk write.  Call CP/M 3.0 BIOS to write one
;       physical sector from buffer.
;
writehst:
```

*(Continued on next page)*

```
        call    rw$init         ; init CP/M 3.0 BIOS
        mvi     a,14            ; write function number
        call    xbios           ; write sector
        sta     erflag
        ret
;
;       Translate sector.  Set CP/M 3.0 track, sector,
;       DMA buffer and DMA bank.
;
rw$init:
        lda     hstsec          ; physical sector number
        mov     l,a
        mvi     h,0
        shld    bcreg           ; sector number in BC
        lhld    xlat            ; address of xlat table
        shld    dereg           ; xlat address in DE
        mvi     a,16            ; sectrn function number
        call    xbios           ; get skewed sector number
        mov     a,l
        sta     actsec          ; actual sector
        shld    bcreg           ; sector number in BC
        mvi     a,11            ; setsec function number
        call    xbios           ; set CP/M 3.0 sector
        lhld    hsttrk          ; physical track number
        shld    bcreg           ; track number in BC
        mvi     a,10            ; settrk function number
        call    xbios
        lxi     h,hstbuf        ; sector buffer
        shld    bcreg           ; buffer address in BC
        mvi     a,12            ; setdma function number
        call    xbios
        mvi     a,1             ; DMA bank number
        sta     areg            ; bank number in A
        mvi     a,28            ; setbnk function number
        call    xbios           ; set DMA bank
        ret
;
;       Print message at HL until null.
;
prmsg:  mov     a,m
        ora     a
        rz
        mov     c,m
        push    h
        call    xconout
        pop     h
        inx     h
        jmp     prmsg
;
;       disk i/o buffer
;
hstbuf: ds      max$sector$size
;
;       variable storage area
;
sekdsk: ds      1               ; logical disk number
```

```
sektrk: ds      2               ; logical track number
seksec: ds      1               ; logical sector number
;
hstdsk: ds      1               ; physical disk number
hsttrk: ds      2               ; physical track number
hstsec: ds      1               ; physical sector number
;
actsec: ds      1               ; skewed physical sector
sekhst: ds      1               ; temp physical sector
hstact: ds      1               ; buffer active flag
hstwrt: ds      1               ; buffer changed flag
;
unacnt: ds      1               ; unallocated sector count
unadsk: ds      1               ; unalloc disk number
unatrk: ds      2               ; unalloc track number
unasec: ds      1               ; unalloc sector number
sekbuf: ds      2               ; logical sector address in buffer
;
spt:    ds      1               ; cpm sectors per track
xlat:   ds      2               ; xlat address
bsm:    ds      1               ; block shift mask
psh:    ds      1               ; physical shift factor
;
erflag: ds      1               ; error reporting
rsflag: ds      1               ; force sector read
readop: ds      1               ; 1 if read operation
rwflag: ds      1               ; physical read flag
wrtype: ds      1               ; write operation type
dmaadr: ds      2               ; last dma address
oldaddr:ds      2               ; address of old BIOS
;
        end
```

End Listing

# Resident System Extensions

## RSX Under CP/M Plus

One of the many new features of CP/M Plus is the ability to use a Resident System Extension (RSX). An RSX allows a programmer to extend or modify any BDOS function, as well as create new ones. To access the BDOS under CP/M, a program loads certain registers with parameters and makes a call to location 0005h. Stored at this location is a jump to the beginning of BDOS. After an RSX has been loaded, the jump instruction points instead to the beginning of the RSX. This allows the RSX to intercept all BDOS calls and to modify them as the programmer sees fit. One or more RSXs can be attached to an executable file using the CP/M Plus system utility GENCOM; they are loaded at the same time as the program.

The applications of an RSX may at first seem limited, but after you have written your first one the uses will seem endless. The idea for my first RSX came about one day while I was playing Pacman on my computer. Every time the little guys moved, the terminal beeped, and there was no way to pause the program if I were interrupted. The answer was simple: I wrote a 37-line RSX that intercepted all console output calls (BDOS functions 2 and 6) and all console input calls (functions 1 and 6).

When the program outputs a character, the RSX compares it with the bell character. If it is a bell, the RSX simply returns to the program; otherwise, it passes the call on to BDOS. When the program makes a console input call, the RSX makes a corresponding console input call and compares the character returned from BDOS with the ESC character. If it is not the ESC character, the RSX returns the character to the calling program. If it is an ESC, the RSX does a console input call using BDOS function 6 (Direct Console I/O) with register E set to 0FDh. This suspends the calling process until a character is typed. Therefore, when you press the ESC key while the program is running, the program is suspended until you press another key.

You could attach this RSX to any other program where you need the ability to temporarily pause the program but it is not available. You could also modify this RSX to redefine any set of keys on the keyboard. RSXs can become much more complex, including routines that monitor a modem port and notify you when someone is trying to access your computer and complete disk or character I/O drivers.

by Garry M. Silvey

*Garry Silvey, Digital Research, P.O. Box 579, 160 Central Avenue, Pacific Grove, CA 93950.*



### RSX Memory Organization

RSXs are loaded when a program containing an RSX is loaded. When the loader detects that the file being loaded contains an RSX, it strips the RSX from the program code, relocates the RSX just below BDOS (or a previously loaded RSX), modifies the jump instruction at location 0005h to point to the beginning of the RSX, and loads the program code beginning at location 0100h in memory. The loader also sets the uninitialized fields in the RSX Prefix.

The figure (page 37) illustrates how memory is configured before and after an RSX has been loaded. Note that the size of the transient program area will shrink as each new RSX is loaded.

If more than one RSX exists in memory, the jump instruction at location 0005h points to the first RSX. The Next field of that RSX points to the next RSX, and so on until the chain of RSXs reaches BDOS.

### Format of an RSX

The first 27 bytes of an RSX contain a data structure called the RSX Prefix that is used by both the RSX and the loader. In the RSX Prefix is a flag that indicates whether the RSX should be loaded in nonbanked systems only and whether or not the RSX should be removed during a warm boot. If the RSX is not removed at warm boot, it will be active until the system is cold-booted; all programs that are executed after the RSX has been loaded will have their BDOS calls intercepted by the RSX.

An example of an RSX that loads only in nonbanked systems is the DIRLBL.RSX file that is included with CP/M Plus. This RSX, attached to SET.COM, modifies the way the directory label is updated in nonbanked systems. If SET is run on a banked system, the RSX is not loaded. The table (page 37) contains a listing of all of the fields contained in the RSX Prefix. Listing One (page 40) shows how an RSX Prefix would be set up in assembly language.

The fields that the programmer must initialize are the Start, Remove, Nonbanked, and Name fields, as well as part of the Next field. The loader will fill in the other fields when it loads the RSX into memory.

The first byte of the Next field should be initialized to 0c3h, the hexadecimal code for the jump instruction. The Start field should be initialized with a jump instruction to the beginning of the RSX code. This is where a function number is tested to see if the RSX should intercept it.

The Remove flag should be set to 0ffh if the RSX is to be removed from memory by the next call to BDOS function 59, Load Overlay. If this flag is set to zero, the RSX will remain in memory and be functional until the system is cold-booted.

Note that the CCP always calls the Load Overlay function during a warm boot.

The Nonbanked flag is set to 0ffh to tell the loader to load the RSX in nonbanked systems only. If the flag is set to zero, the RSX is loaded in both banked and nonbanked systems. The Name field should be initialized to the name of the RSX; the name must be 8 bytes long, so pad it with blanks if necessary.

### Function of an RSX

Once a program that contains an RSX has been loaded, the RSX intercepts all BDOS calls (Call 0005h) before they reach BDOS. The RSX then determines if the function number being passed to BDOS matches a function number that it should extend, modify, or create. If the function is not relevant to the RSX, the call is passed up the RSX chain by jumping to the Next field of the RSX Prefix.

If the RSX needs to make a BDOS call, it must call the address stored in the Next field instead of the address at location 0005h. By using the address in the Next field, the call starts with the next RSX or BDOS. If the RSX makes a BDOS call using location 0005h, the call will eventually reach the same RSX, which could result in an infinite loop!

The actions that an RSX can take are many. The RSX may simply modify the parameters passed to it, then let the call continue on to BDOS; in this case, the last instruction of the RSX would be a JMP NEXT. The RSX may catch the call, perform a number of BDOS calls using the instruction CALL NEXT, and then either return to the transient program by using a RET instruction or pass the original call on to BDOS by using a JMP NEXT instruction. The RSX may also catch a BDOS call, perform the BDOS action (i.e., direct I/O port accessing) without ever using BDOS, and then return to the calling program.

The RSX should set up a local stack that is large enough for its own needs. If it does set up its own stack, it must restore the stack pointer to the entry stack before exiting.

### RSX Example

The RSX example included with this article will kill two birds with one stone. It makes a good example for how to use RSX, and it "patches" an incompatibility between CP/M 2.2 and CP/M Plus.

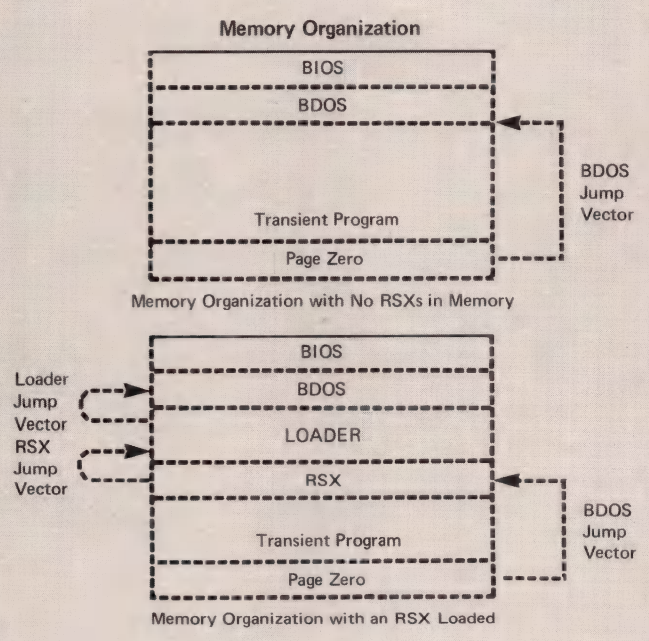
In CP/M 2.2 the filename specified in the FCB given for BDOS function 15, Open File, can contain question marks; the BDOS will open the first file whose name matches the ambiguous filename. If the same program is run under CP/M Plus, the BDOS error "? in filename" occurs and an error is returned. If a program you have developed under 2.2 calls the Open File function with question marks in the FCB, you simple attach the following RSX to it, and it will run under CP/M Plus as expected.

The sample RSX modifies the BDOS Open File function with the following routine (see Listing Two, page 40). When BDOS is called with the function number equal to 15, the RSX intercepts the call and checks the FCB for question marks. If no question marks appear in the FCB, the RSX passes the call on to BDOS for normal processing. If there are question marks in the FCB, the RSX uses the ambiguous filename in the FCB in a call to BDOS function 17, Search

| | |
|---|---|
| **Serial Number:** | *Holds the serial number of the operating system.* |
| **Start*:** | *Contains a jump instruction to the beginning of the* RSX *code.* |
| **Next:** | *Contains a jump instruction to the next RSX in the* RSX *chain or to* BDOS. |
| **Previous:** | *Contains the address of the previous (newer)* RSX *or location 5 if this is the first* RSX. |
| **Remove*:** | *This flag tells the loader whether or not the* RSX *should be removed during the next call to the loader via* BDOS *function 59, Load Overlay. (This function is always called during warm boot.)* |
| **Nonbanked*:** | *Tells the loader if the* RSX *should be loaded in nonbanked systems only.* |
| **Name*:** | *Contains an 8-character name for the* RSX. |
| **Loader:** | *This flag is set if the* RSX *is the last one in the* RSX *chain. If the flag is set, it indicates that the* RSX *is actually the loader.* |
| **Reserved:** | *This field is reserved for use by* BDOS. |

**The programmer must initialize this field.*

**Table**
**RSX Prefix Fields**




**Figure**

First. If the search returns an error, the RSX returns a "file not found" error to the program. If the search is successful, the RSX takes the filename found and places it into the user's FCB. It then jumps to BDOS with register C set to the Open File function. BDOS then opens the file as expected and returns to the program.

Also included with this article is a short program that tests the functionality of the RSX example (Listing Three, page 43). It prompts the user for a filename, uses BDOS function 152, Parse Filename, to set up an FCB, then calls BDOS function 15, Open File. If the RSX has been attached, and the user gives an ambiguous filename when prompted, the first file whose name matches the filename is opened as it would under CP/M 2.2. If the RSX has not been attached, an error is returned.

To use the RSX in Listing Two, first type it in using your editor, then perform the following steps:

(1) Use RMAC to assemble the file.

(2) Use LINK to link the file. Use the OP option so LINK will produce a Page Relocatable (.PRL) file.

(3) RENAME the file created by LINK so that it will have a filetype of .RSX.

(4) Use the test program supplied, or a program that you know uses FCBs with question marks, and attach the RSX file to it using GENCOM. ■■J

**(Listings begin on page 40)**



## Example

```
13A>RMAC OPENRSX
CP/M RMAC ASSEM 1.1
00C7
001H USE FACTOR
END OF ASSEMBLY

13A>LINK OPENRSX [OP]
LINK 1.31
TRUE     FFFF   FALSE    0000   PRINTS   0009   OPENFI   000F
SEARCH   0011   SCB      0031   CALLRS   003C   DEBUG    FFFF
NEXT     0109   PREV     010C   REMOVE   010E   NONBNK   010F
RSXNAM   0110   START    011B   CAUGHT   0132   LOOP     0141
DOIT     0155   FOUND    0170   LDIR     0189   SCBPB    019A
OFFSET   019A   OPERAT   019B   VALUE    019C   USERST   019E
RSXSTA   01AA   QMESS    01AA

ABSOLUTE              0000
CODE SIZE    00C7     (0100-01C6)
DATA SIZE             0000
COMMON SIZE           0000
USE FACTOR            04

13A>REN OPENRSX.RSX=OPENRSX.PRL
13A>MAC OPEN
CP/M MACRO ASSEM 2.0
0233
001H USE FACTOR
END OF ASSEMBLY

13A>HEXCOM OPEN
HEXCOM VERS: 3.00

FIRST ADDRESS      0100
LAST ADDRESS       0232
BYTES READ         0101
RECORDS WRITTEN    03

13A>GENCOM OPEN OPENRSX

GENCOM completed.

13A>
```

**The RSX assembly language file (Listing One) is called OPENRSX.ASM, and the test program (Listing Three) is called OPEN.ASM.**

# NO HYPE

## JUST FACTS

In October 1983 The National Software Show became the headline event of the microsoftware industry: the first international microsoftware-specific tradeshow for ISOs, OEMs, Systems Houses, and Volume Buyers for large corporations.

Once again, the second annual National Software Show gives attendees all the latest-breaking information about microsoftware, before your customers ask for it.

### THE UNIX MOVEMENT

The UNIX System is moving the industry out of its infancy and into adulthood. How does UNIX affect the reseller, corporate user, or Systems House/OEM? What's going on with UNIX? Where is the best place to find the answers? . . . At The National Software Show.

### RIGHT TIMING

Smart ISOs are kicking-off their buying season this September at The National Software Show. Preview and order new programs before the year-end rush.

### ALL UNDER ONE ROOF

Virtually every major microsoftware producer in over 700 exhibit spaces will share their expertise and demonstrate their top products. All on one level. All easy to find.

NSS

### DEALER FOCUS

A comprehensive conference schedule targets ISOs. Know the answers before they affect your bottom line. Included in an Exhibits-Only admission: three keynote panels, and daily press conference schedule where industry leaders will announce and demonstrate new products. Keynote topics are: 'Multi-User System Update,' 'The UNIX System Marketplace,' and 'The Micro to Mainframe Link.'

### DON'T FALL BEHIND

Open your most important selling season by attending The National Software Show. Learn the inside information on microsoftware, before everyone else does. Register now!

3-Day Exhibits/Full Conference - $125
3-Day Exhibits and One Day Conference - $65
3-Day Exhibits Only - $25

### CALL TODAY

Sales representatives are on hand at 800-732-2300 (outside CA) or 415-924-1194 ( inside CA) to answer any questions and rush you a NSS PreShow Information Guide, listing conferences and speakers in detail. We accept American Express.

# The National Software Show™

## September 5-7, 1984

### Anaheim Convention Center • Anaheim, California

Produced by Raging Bear Productions, Inc. 21 Tamal Vista, Suite 175, Corte Madera, CA • (415) 924-1194 or (800) 732-2300

# RSX Under CP/M (Text begins on page 36)

## Listing One

RSX Prefix Assembly Language Example

```
serial:   db   0,0,0,0,0,0     ; The loader will put the
                               ;    serial number here.
start:    jmp  RSXstart        ; Jump to start of the
                               ;    RSX code.
next:     jmp  $-$             ; jmp instruction
                               ; Address of the next
                               ;    module, this is
                               ;    filled in by the
                               ;    loader.
prev:     dw   0               ; The loader will put the
                               ;    address of the
                               ;    previous module here
remove:   db   0ffh            ; Remove flag
nonbank:  db   0               ; Nonbank flag
name:     db   'RSX NAME'      ; Name of the RSX
Loader:   db   0               ; Loader flag
          db   0,0             ; Reserved for system use

RSXstart:                      ; Actual RSX code starts here
```

End Listing One

## Listing Two

Open File RSX

```
;
; This program is assembled using RMAC and LINK to create an RSX to
; catch BDOS open file calls which specify an FCB that has question
; marks in it.  If there are no ?'s in the FCB, the call is passed on
; to BDOS.  If there are ?'s, this RSX will use the FCB pointed to by
; register pair DE to call the BDOS search first function.  If the
; search is not successful, it will return the appropriate error to
; the calling program.  If the search was successfull, the filename
; found is transfered from the DMA buffer to the FCB address that was
; originally passed in register DE.  The RSX will then execute a normal
; open file call.
;

TRUE    equ     0ffffh
FALSE   equ     NOT TRUE

PRINTSTR        equ     09              ; Print String
OPENFILE        equ     15              ; Open File
SEARCHF         equ     17              ; Search for First
SCB             equ     49              ; Get/Set System Control Block
CALLRSX         equ     60              ; Call RSX

DEBUG   equ     TRUE                    ; will display more messages
                                        ; while debuging
```

```
; Start of the RSX Prefix
        db      0,0,0,0,0,0             ; serial number
        jmp     START                   ; start of this RSX
NEXT    jmp     $-$                     ; jmp instruction
PREV    dw      0                       ; address of previous module
REMOVE  db      0ffh                    ; remove on warm boot
NONBNK  db      0                       ; load for banked and non-banked
RSXNAME db      'OPEN    '              ; name of RSX
        db      0,0,0                   ; used by BDOS

; start of RSX code
START:  mov     a,c                     ; check BDOS function number
        cpi     OPENFILE                ; is it an open file call ?
        jz      CAUGHT                  ; if so, process it
; If DEBUG is true, then check the function number for the BDOS function
; Call RSX.  If the function number matches, return to the calling program
; with register A set to 0 to tell the program that the RSX has been loaded.
if DEBUG
        cpi     CALLRSX                 ; is it a call to this RSX ?
        jnz     NEXT                    ; nope, pass to next RSX or BDOS
        mov     a,d                     ; Is it the test program calling ?
        cpi     0ffh
        jnz     NEXT
        cmp     e                       ; register DE equal 0ffffh ?
        jnz     NEXT                    ; nope, pass call on to next module
        xra     a                       ; otherwise zero A
        ret                             ; return to calling program
else
        jmp     NEXT                    ; jump to next module if not open call
endif

CAUGHT: lxi     h,0
        dad     sp                      ; get current stack pointer
        shld    USERSTACK               ;       and save it
        lxi     sp,RSXSTACK             ; use local stack
        push    b                       ; save users registers
        push    d

;       check filename for ?'s
        mvi     b,11                    ; max # of characters in a filename
        inx     d                       ; point to filename in FCB
LOOP:   ldax    d
        cpi     '?'                     ; is it a ?
        jz      DOIT                    ; found a ?, so execute rest of RSX
        inx     d                       ; try next character
        dcr     b                       ; decrement counter
        jnz     LOOP

;       If the program has reached here, there are no ?'s in the filename
;       so pass the call on to BDOS (or the next RSX) for normal execution.

        pop     d                       ; restore the users registers
        pop     b
        lhld    USERSTACK
        sphl                            ; restore users stack
        jmp     NEXT                    ; pass to next RSX or BDOS

DOIT:   ; Here a Search First function is executed, then the FCB is
        ; updated with the filename that was found, and then a Open
        ; File command is sent to BDOS.  If the search first call
        ; returns an error, the RSX returns the appropriate error.

if DEBUG
        ; Tell the user we found a "?"
        lxi     d,QMESS
```

*(Continued on next page)*

## Listing Two

```
        mvi     c,PRINTSTR
        call    NEXT
endif
        pop     d                       ; get the FCB address
        push    d                       ; save it again
        mvi     c,SEARCHF               ; prepare for a Search First call
        call    NEXT                    ; (call BDOS)
        cpi     0ffh                    ; not found ?
        jnz     FOUND                   ; continue processing
        ; An error occured during the Search First call. In this
        ; case, we do not want to pass the original call onto BDOS since
        ; there are still question marks in the FCB.  Instead, return to
        ; the calling program with errors in a,hl as set by BDOS.
        pop     d                       ; restore users registers
        pop     b
        lhld    USERSTACK
        sphl                            ; restore users stack
        ret                             ; return to users program

FOUND:  ; The Search First was successfull, so update the users FCB
        ; with the FCB of the file that was found, and jump to BDOS's
        ; Open File function.

        ; First multiply directory code by 32 to get offset to the
        ; FCB of the file that was found.
        add a  !  add a
        add a  !  add a
        add a
        push    a                       ; save offset

        ; The value of the current DMA address must be determined so
        ; the RSX can retrieve the FCB of the file that was just found.
        ; The calling program may have modified the current DMA address,
        ; so the DMA address will have to be extracted from the SCB.

        mvi     c,SCB                   ; get/set SCB function
        lxi     d,SCBPB                 ; SCB pb address
        call    NEXT                    ; call BDOS
;       Register pair HL now contain the DMA address
        pop     a                       ; restore the offset to the FCB
        mov     c,a
        mvi     b,0                     ; put offset into bc
        dad     b                       ; hl now points to the FCB of the file
                                        ; that was just found
        inx     h                       ; point hl to start of file name
        pop     d                       ; get d (which points to the users FCB)
        push    d                       ; save d again
        inx     d                       ; point to file name
        ; Now the calling program's FCB must be updated with the FCB of
        ; the filename that was just found.  The filename field of the
        ; FCB is the only part that needs to be updated.
        ;
        ; If you have a Z80 CPU, use the following code.
;       lxi     b,11
;       dw      0b0edh                  ; Z80 ldir instruction
;
        ; If you have a 8080 CPU, use the following code.
        mvi     b,11
LDIR:   mov     a,m ! inx h
        stax    d   ! inx d
```

```
        dcr     b
        jnz     LDIR

        ; The calling program's FCB now has a full filename in it, so
        ; now the RSX can setup an Open File call.
        pop     d
        pop     b                       ; Restore users registers.  Register
                                        ; C will be restored with the
                                        ; original Open File function
                                        ; number in it.
        lhld    USERSTACK
        sphl                            ; restore users stack
        jmp     NEXT                    ; BDOS will now open the file normally

SCBPB:  ; SCB parameter block: Set up to Get the DMA address stored
        ; in the SCB at offset Ø3ch.
OFFSET:         db      Ø3ch                    ; offset in SCB to DMA address
OPERATION:      db      Ø                       ; get operation
VALUE:          dw      Ø

USERSTACK:      dw      Ø                       ; storage for users stack

                ds      1Ø                      ; space for stack
RSXSTACK:

        ; If debug is true, include the following message.
if DEBUG
QMESS   db      '? in FCB, caught by RSX...',Ødh,Øah,'$'
endif
```

**(End Listing Two)**

## Listing Three

```
                Open File RSX Test Program


;
; Sample program to test the RSX in listing #2.  The program asks
; for a filename and builds and FCB for the file using the BDOS
; Parse Filename function.  It then tries to open the file.
;
; If the filename has any question marks in it, BDOS would normally
; terminate the program and display an error message.  After the RSX
; is installed, the file is opened as under CP/M 2.2 and no error is
; returned.
;

; miscellaneous equates
TRUE    equ     Øffffh
FALSE   equ     not TRUE
BDOS    equ     5                       ; BDOS jump address
CR      equ     13                      ; carriage return
LF      equ     1Ø                      ; line feed

PRINTSTR        equ     Ø9              ; Print String
RBUFF           equ     1Ø              ; Read Console Buffer
OPENFILE        equ     15              ; Open File
SEARCHF         equ     17              ; Search for First
SCB             equ     49              ; Get/Set SCB
CALLRSX         equ     6Ø              ; Call RSX
```

*(Continued on next page)*

## Listing Three

```
PARSE           equ     152             ; Parse Filename

DEBUG   equ     TRUE

        org     0100h

START:  ; print signon message
        lxi     d,SIGNON
        mvi     c,PRINTSTR
        call    BDOS

if DEBUG
        ; See if the RSX has been attached...
        mvi     c,CALLRSX
        ; Normally, register pair DE would point to a RSX parameter
        ; block, but since the RSX will not use any parameters, load
        ; DE with a value of 0ffffh to differentiate this RSX call from
        ; other RSX calls that could be used by other programs.
        lxi     d,0ffffh
        call    BDOS
        cpi     0                       ; is there a RSX attached ???
        jz      PRSXM                   ; yes, tell user
        mvi     c,PRINTSTR
        lxi     d,NO                    ; tell the user there is no RSX
        call    BDOS
        ; Print RSX message
PRSXM   mvi     c,PRINTSTR
        lxi     d,RSXM
        call    BDOS
endif

        ; Prompt user for filename
        mvi     c,PRINTSTR
        lxi     d,PROMPT
        call    BDOS

        ; Get the filename from the user
        lxi     d,INBUFF
        mvi     c,RBUFF
        call    BDOS

        ; Move the cursor to the next line
        lxi     d,RETURN
        mvi     c,PRINTSTR
        call    BDOS

        ; Set up the Parse Filename Control Block and parse the filename
        lxi     d,PFCB
        mvi     c,PARSE                 ; BDOS parse file name function
        call    BDOS

        ; The parsed filename is now in the FCB, so now open it
        lxi     d,FCB
        mvi     c,OPENFILE              ; BDOS open file function
        call    BDOS
        cpi     0ffh                    ; error returned ?
        jnz     FOUND                   ; if not exit program
```

```
        ; Print an error message if the file was not found
        lxi     d,ERRMESS               ; print error message
        mvi     c,PRINTSTR              ; BDOS print string function
        call    BDOS
EXIT:   ; Return control back to the CCP
        mvi     c,0                     ; BDOS warm boot function
        jmp     BDOS

FOUND:  ; Tell the user what the name of the file that was found is
        lxi     d,ENDMESS
        mvi     c,PRINTSTR
        call    BDOS                    ; print ending message
        lxi     d,FCB+12                ; point de to end of filename
        mvi     a,'$'
        stax    d                       ; put a $ there
        lxi     d,FCB+1                 ; point d to first character of name
        mvi     c,PRINTSTR
        call    BDOS                    ; print the filename
        lxi     d,RETURN                ; drop to next line
        mvi     c,PRINTSTR
        call    BDOS
        jmp     EXIT                    ; return to ccp

PFCB:   ; Parse Filename Control Block
        dw      INBUFF+2                ; address of input string
        dw      FCB                     ; address of target FCB
INBUFF: ; Console input buffer
        db      14,0                    ; number of characters to read
        ds      14                      ; input buffer space
FCB:    ds      36                      ; FCB location

; Terminal messages
ERRMESS:db      'file not found',cr,lf,'$'
SIGNON: db      'open file RSX test program...',cr,lf,'$'
ENDMESS:db      'name of file found : $'
RETURN: db      cr,lf,'$'
RSXM:   db      'RSX has been installed.',cr,lf,'$'
PROMPT: db      'enter file name : $'
NO:     db      'no $'
        end
```

**End Listings**

---

# p–A Small C Preprocessor

*Last month Dr. Schreiner provided us with* cc, *a driver for a Small C programming system. This month he presents* p, *the preprocessor for Small C referred to in the previous article.* p *provides yet another tool for greater convenience and power in the Small C environment. Also included this month is code for adding random order release of memory to the routine library. While not included with* cc *last month, it is applicable in that context as well.—Ed.*

Jim Hendrix's Small C compiler (*DDJ*, December 1982) supports some of the preprocessor commands that are usually available in C systems: a symbolic name can be defined for an arbitrary text, which will then be inserted whenever the name appears in the source program. This facility is most frequently used to give meaningful names to various important constants in a program, but it can also be used to give C programs the appearance (almost) of Pascal programs, to substitute one function name for another, and so on. Hendrix's compiler also supports one level of file inclusion, although with a somewhat nonstandard syntax, and it supports conditional compilation based on whether or not certain symbolic names have been defined.

Such a preprocessor is an important tool in its own right. It can be combined with other language processors and assemblers as well. It also becomes considerably more flexible if text substitution can be parameterized (i.e., if macro calls can have arguments and if file inclusion is performed to arbitrary depth). If the preprocessor eliminates C-style comments and translates C-style constants to decimal notation, it simplifies the compiler's job significantly and at the same time obtains a uniform approach to constant notation, commenting conventions, and file inclusion.

*p*, the program described in this article (and presented in the Listing One on page 52), is such an independent preprocessor. It is written in Small C and supports all the features of the regular C preprocessor described by Kernighan and Ritchie (*The C Programming Language*, Prentice Hall, 1978) with the exception of an *if* preprocessor statement with a constant expression argument.

by Axel T. Schreiner

*Axel T. Schreiner, Universität Ulm, Sektion Meth.d.Informatik, Oberer Eselsberg, 7900 Ulm (Doran) West Germany.*

**Unix is a trademark of Bell Laboratories.*

## Features

*p* is based on a runtime support that passes arguments to the main program; it expects to be called as follows:

*p* (option) . . . (inputfile (outputfile))

If no files are explicitly specified, *p* will read from "standard input" and write to "standard output" (the runtime support presumably makes those connections as well). If the runtime support would normally connect to the terminal, it is quite simple to test certain features through input from the terminal or to see the results of a preprocessor run directly at the terminal.

You may specify options in any order. They are generally cumulative. The following options are available:

| | |
|---|---|
| *-d* name (=value) | Define a symbolic name |
| *-e* | Suppress position stamps |
| *-i* drive | Search for file inclusion |
| *-u* name | Undefine a symbolic name |
| *-v* | Verbose — for debugging |

You may redefine symbolic names when *p* is called. This is quite convenient for maintaining various versions of a program. *p* predefines the name cpm. You can undo this (in particular) using the option to undefine a name. To permit a compiler to emit error messages referencing the original source files, *p* will create a position stamp (i.e., a line starting in "#" and containing a decimal line number and, if known, a filename whenever this is necessary for correct sequencing of output lines. Hendrix's Small C compiler cannot handle these position stamps; therefore, they can be suppressed.

CP/M identifies files by name and disk drive. *p* therefore will search *include* files on various disks: first on the disk of the input file (which may be the currently selected disk) and last on the currently selected disk. In between, *p* may search other disk drives optionally, if appropriate options have been specified. The search proceeds from right to left over the *-i* options.

The options are clearly patterned after the Unix* system. The main program expects to receive pointers to the various options as a vector "argv." The number of such options, including (theoretically) the program name as first option, is also passed as an integer "argo." *p* is tolerant enough to accept the parameter of an option (such as an *include* drive) as part of the option or as a separate, immediately following option. Options, however, must precede the filenames.

Once started, *p* will read the input file and write the output file. C-style comments (i.e., arbitrary texts enclosed in /* and */ sequences) are replaced by single blanks. Next preprocessor command lines are processed. Then in regular text lines macro calls (i.e., appearances of defined names possibly followed by a list of argu-

ments in parentheses) are replaced. The replacement text is surrounded by blanks and is reprocessed for further macro calls. Recursion may happen, but reprocessing is aborted after a few attempts with an appropriate message.

Input lines, as well as output lines, may be arbitrarily long. You may continue input lines over several source lines by placing a backslash right before the end of the source line to be continued. C is free format, but preprocessing is line oriented; continuation should be necessary (and come in handy) only for preprocessor command lines, macro calls, and very long strings. Macro calls and strings must be fully contained within one (possibly continued) input line.

A macro call is not recognized within a comment, a preprocessor command line, a string, or a character constant. The replacement text is surrounded by blanks. You cannot use two adjacent macro calls to create another macro call from their replacement text.

If you define a macro with parameters, the macro call consists of the macro name and a list of arguments, separated by commas and enclosed by parentheses. While in the definition the left parenthesis must immediately follow the macro name, it need not in the call. The call, however, must be completely contained on one input line to avoid rather sticky questions should macro be redefined inside its call, etc.

*define* name replacement-text
*define* name(name,. . .) replacement-text

*undef* name

*ifdef* name
*ifndef* name
*else*
*endif*

*include* "filename"
*include* (filename)

*line* linenumber filename

*define* is used to define a symbolic name for an arbitrary replacement text. You may parameterize the replacement text. The parameter names are local to each definition, and unfortunately no test exists to determine whether two parameter names within one definition are one and the same. Names follow C conventions; i.e., they must start in a letter or underscore, can contain letters, digits, or underscores, and can be arbitrarily long.

Redefinition is possible but provides a message, *undef* can be used to remove a definition, and there is no complaint if the relevant name was never defined.

*ifdef* is fulfilled if the specified name is currently defined; *ifndef* is fulfilled if it is not. In either case, subsequent input lines (including preprocessor commands) are processed or skipped depending on the condition. *else* reverses the current value of the condition, and *endif* terminates the construct. You may nest these constructs to any depth. *else* should only be used once per construct, but this is not checked. Each *else* reverses the current state of things.

*include* causes file inclusion. The filename should follow CP/M conventions. If it does not contain an explicit disk drive specification, the file is searched on the list of drives beginning with the input file drive and ending with the currently selected drive. If the filename is enclosed in angle brackets rather than double quotes, the first drive on the list is skipped. (Normally, brackets are used to designate public *include* files, presumably residing on the currently selected disk and not on the disk of the input file.)

A macro argument is an arbitrary text and may even contain a comma within balanced parentheses, a string, or a character constant. The text is substituted wherever the corresponding parameter appears in the macro definition, even within a string. No blanks are forced around the argument text. There must be exactly as many arguments as there are defined parameters.

*p* removes leading white space from the output lines and converts constants to decimal notation. These "features" can be removed easily. They do impair the general applicability of the program, but they overcome certain problems in Hendrix's compiler while significantly shortening the output file.

The following constant notations are accepted and are converted to (signed) decimal representation:

| | |
|---|---|
| digits | decimal constant |
| 0 digits | octal constant(digits 0 . . 7) |
| 0x digits | hexadecimal constant |
| 0X digits | (digits 0 . . 9, a . . f, A . . F) |
| 'c' | character constant |

Clearly, no blank may separate the base prefix from the actual constant. Character constants may be escaped; escape sequences consist of a backslash character followed by other characters. The following escapes are recognized:

| | |
|---|---|
| b | backspace |
| f | form feed |
| n | newline (line feed) |
| r | return |
| t | tab |
| ' | single quote |
| " | double quote |
| \ | backslash itself |
| d | octal, up to three digits 0 . . 7 |

Character constants are converted to decimal notation. This is reasonable for C programs, but it might cause problems elsewhere.

Preprocessor command lines begin with a "#" symbol. White space may follow, and then a keyword must be distinguishable. Depending on the keyword, there may be parameters; the rest of the command line can be quite arbitrary. The following commands are supported:

You may nest *include* to any depth, assuming the runtime support for this program is reasonable. Once the end of an *included* file is reached, processing continues after the *include* command causing the file inclusion. If a file cannot be opened or found, a message is printed, but processing continues.

The *line* command is intended for program generators. For purposes of diagnostics and position stamps in the output file, *p* will accept a line number and optionally a filename from the *line* command.

Other lines starting in "#" are processed as if they were text. *p* will thus pass through such things as "asm" and "endasm," provided these lines are not modified by macro expansions. Position stamps from a previous run of *p*

would also be passed through again.

**Implementation**

The task of preprocessing can be nicely structured into four problems, each solved essentially by a single C function:

```
take care of start options, initialize

while (there is another line)

    if (after removing comments, there
    is something && 

        after observing commands, there
        is something)

            preprocess the text line
```

main() takes care of start options and initialization. getline() collects a nonblank line and takes care of line continuation. comment() eliminates comments, which may extend over several lines, and removes leading white space. command() knows whether you are currently skipping in observance of some *if* construct; if you are, command() pretends that a command was actually found so that the input line is not processed further. If there is a preprocessor command, command() will recognize and execute it. If the current line is regular text, process() takes care of macro expansion and output.

getline() simply collects input characters into a buffer until a newline character or end of file is found. If there is a backslash followed by a newline, both characters are ignored (but source lines are counted). If there is end of file following a backslash, getline() complains. At end of file, getline() attempts to pop the stack of open input files. getline() returns once it encounters a nonblank input line or if the end of the initial input file is reached. Non-ASCII characters are not accepted as input to simplify subsequent processing.

For input and output two buffers must be maintained and should be able to grow more or less without limit. The rebuff() routine, which is called with a full buffer, handles this. It will relocate the buffer and copy the information over. Clearly, no other pointers into the buffer should exist at that point.

The comment() algorithm is quite simple: it copies everything until /* then quits copying (and reporting that there is text material) until */ is found. Matters are complicated slightly by strings, (invalid) character constants, and backslashes. The problem is handled with the state variable "cmode," which maintains the current context — comment, string, character constant — across calls. comment() will suppress blank lines, as well as initial white space.

command() deals with preprocessor commands in all those line buffers that comment() did not prevent from being passed on. Calls to the symbol table management routines handle command processing. *if* constructs are implemented through two variables: iflevel, which counts the current nesting depth of these constructs, and skip. skip is usually and initially zero, indicating that text should be processed. If text should be ignored (due to some *if* or *else*), skip is set to the value of iflevel at which skipping should terminate. If you are skipping and reach *else* or *endif* at the proper level, you are done; skip reverts to zero.

There really should be only one *else* per *if*. Enforcing this would require a stack to indicate at each "iflevel" if we have already seen *else* or not. I felt that this was not really necessary — consequently (as in Hendrix's compiler), multiple *else* are allowed.

*line* and *include* require a certain amount of data processing. The syntax must be verified, and the relevant information must be stacked. There are stacks of open file pointers, filenames, and last line numbers. All the stacks are handled by push routines, which return the address of a new element linked before the stack. The result of push, therefore, must always be assigned back to the stack top pointer, which is passed as an argument. A common pop() routine handles removal of unwanted elements. A string stack is also used to hold the *include* prefixes passed as options.

process() does the actual work if a line is ever passed on to it. The line buffer is scanned and copied to the oline buffer. The outmacro(), outnumber(), outdelim(), and out() routines are called to manage processing of a macro call, a numerical constant, a string or character constant, and a simple character, respectively. As long as a macro was actually expanded in a pass over line, the two buffers are interchanged and processing is repeated until either no more expansion is performed or a count runs out (to prevent infinite recursion). output() takes care of emitting position stamps and the processed text.

Finding names in the symbol table and undefining them is quite simple: the symbol table is a one-way linked list of entries, each containing a pointer to the string defining the symbol and the symbol name itself. If the symbol is a parameterized macro, the name is followed by a left parenthesis and a (binary) parameter count. find() must take care to match the names correctly. If a symbol definition is parameterized, the value contains (binary) parameter numbers in place of the original parameter names. Each parameter number takes just one byte and is flagged in the high bit since all other text is ASCII.

The symbol table can employ a hashing mechanism. If the symbolic name HASH is undefined, the symbol table consists of a single linear list. If the name is defined, it must be as a power of 2, designating how many such linear lists should be kept. In this case each name is first converted to a number indicating which of the lists should be used. This simple mechanism and a rather naive hash function in find() cut preprocessing time by about 20 percent.

define() does a significant amount of text processing to prepare a parameterized macro definition. marknm() separates the macro header from the macro value within the line buffer. ismacro() and isname() make sure that the macro header uses only appropriate symbols and the proper arrangements of commas and parentheses. ismacro() prepares yet another string stack of parameter names. define() then removes leading and trailing white space from the value and replaces parameter names by flagged numbers. Finally you must guard against redefinitions and store the result. A redefinition is flagged only if it is truly different — therefore, header files usually can be read several times without complaint.

outmacro() also does a large amount

# Don't call her cheap. Call her beautiful.

# The Bonnie Blue™

## Word Processing System for the IBM Personal Computer

It's obvious what makes her so cheap, but what makes Bonnie Blue so beautiful? Bonnie Blue is a new and easy-to-use word processing program for the IBM Personal Computer.

**The Full System.** The Bonnie Blue System includes in one program a full screen Editor, a Printing module and a useful Toolbox. It includes the features you've come to expect, and more:

complete cursor control: by character, word, line; page up and down instantly; go to top, bottom of document; auto scroll towards top or bottom
word wrap
margin justification, centering
adjustable margins, tabs, indents
reformat paragraphs
move, copy, delete, paste blocks
find with delete, insert, replace and wild card characters
keyboard remapping
multi-line headers, footers

Bonnie Blue can handle lines longer than the screen is wide, by horizontally scrolling the line. And, unlike some programs, Bonnie Blue lets you include any displayable character in your text, such as block graphics and foreign language characters.

**Unique Features.** With Bonnie Blue, you can "paint" display attributes onto your text, by the character, word, or line, or automatically as you enter text. With the monochrome adapter, you can paint any combination of underlined, bold, reverse video or blinking. With an 80 column monitor and the color/graphics adapter, this translates into a palette of 16 color combinations to choose from. And if your computer has both monitors, Bonnie Blue lets you use them both, shifting back and forth as you wish.

**Powerful Printing Module.** You can use these colors or display attributes to highlight text on the screen, and Bonnie Blue can remove them from a file when you want (all files created by Bonnie Blue are DOS standard). The Printing module understands these text attributes, and you can map them into any single printer function or combination.

For example, normally you would want underlined text to print underlined. But you can tell Bonnie Blue to print underlined characters as both underlined and bold. Bright text on the screen can mean double struck, or emphasized and in italics. You are at the controls.

The first Print formatting module supports all the text capabilities of the Epson MX series with Graftrax Plus. By the time this ad appears, we will be supporting other popular dot-matrix and letter quality printers.

More than thirty "dot" commands give you added control of the format of your finished document. You can send it to a disk file instead of the printer, or preview the final page formatting on the screen.

**Toolbox.** The Toolbox is a set of useful functions, called "filters" that allow you to extract information from your files and transform their content. With these tools, you can join files together, sort lines of text, count words, find and substitute patterns, etc. Writers and programmers find this a useful collection of productivity enhancers.

Bonnie Blue is also great for a hard disk system. A thorough User's Guide, complemented by help screens and roadmaps, make the Bonnie Blue an exceptionally easy-to-learn and easy-to-use system.

Order yours today, or send for our free brochure. Bonnie Blue is available exclusively from **Bonnie Blue Software,** P.O. Box 536, Liverpool, NY 13088.

IBM Personal Computer is a trademark of IBM Corp. Epson Graftrax Plus is a trademark of Epson America Inc.

**Only $50**

## Bonnie Blue Software

Post Office Box 536
Liverpool, NY 13088

☐ Send me the Bonnie Blue System. I am enclosing $50 (NY State residents please add 7% sales tax).

☐ Please send literature. I have a _____

☐ Check enclosed ☐ VISA ☐ MasterCard Sorry, no COD.

Credit Card No._____ Expires_____

Signature_____

Name_____

Address_____

City_____ State_____ Zip_____

Company_____

**Minimum recommended system:**
IBM PC, 128K, 2 disk drives, PC-DOS 1.1 or 2.0, 80-column monitor or monochrome adapter, or both, Epson MX-80 or MX-100 with Graftrax Plus.

*Versions available soon for PCjr. Write for details.*

184

Circle **no. 10** on reader service card.

of text processing. If the macro is parameterized, markarg() is used to flag the text arguments in the input buffer and to prepare a stack of pointers to these argument values. This task, too, is complicated by the usual potential assortment of parentheses and string delimiters. Once the argument list is collected, it is a simple matter (possibly handled with outarg()) to emit the macro definition with the argument texts in place.

### Installation

Getting *p* to run on your system might be a bit tricky. There is a bootstrap problem — *p* uses features supported only by *p* and not by Hendrix's compiler — and there is a problem concerning the runtime support.

Overcoming the bootstrap problem is actually quite simple. You should replace double quotes as a character constant by the value 34 (my Small C compiler got confused in certain places until I did this). You need to replace all constants that your compiler does not support (e.g., the definitions of PARM and PARMNO and the values for base in the routine outnumber()). You might have to replace the character constants in the routine outdelim() if your compiler does not yet support those escape sequences. Finally, you will have to play preprocessor for those macros that are parameterized. (Yes, it was not nice to use those, but I did want to show how parameterized macros can be used to clarify data types.)

The runtime support is quite a different matter. While I agree with Jim Hendrix that these matters ought to be standardized, I am much in favor of programming on my CP/M system at home just as I do on our Unix systems at the office. I have actually made a runtime support that looks like the standard libraries available with Unix version 7 and above, is based on CP/M, and supports all BIOS and BDOS calls from Small C.

I have had access to chapter 17 of Hendrix's book on Small C, describing his runtime support. While I did not have access to the runtime support itself and therefore could not test it, I believe that installing *p* should be quite simple. The following probably must be done:

FILE should be defined as *int*.
__drive() needs to access BDOS function 25.
__narg() is supported by the compiler.
index() is Hendrix's strchr().
itod() can be coded using Hendrix itod().

I process the arguments to main() directly. Depending on the actual implementation, you might have to use Hendrix's function getar().

I am assuming that the storage allocator, calloc()/cfree(), supports random order release of memory. The code in Listing Two (page 81) may be useful to those wishing to add such random order release to the Hendrix-Payne library published in the May and June 1984 issues of *DDJ*.

Meanwhile, you probably should consult Kernighan and Ritchie's book to learn about all the routines that are mentioned "extern" at the beginning of the program. Most of them are quite simple to construct. It is essential, however, that you provide multiple open files so that arbitrary nesting of file inclusion is possible. You will also need a memory allocator, calloc() and cfree(), that will reuse available space and is reasonably stable. I am using a scheme where memory above the load module and below the stack is managed by a list of words, each word pointing to the next. The low bit in each such word indicates if the area past that word and up to the next one is allocated or free. Kernighan and Ritchie mention, and Unix supports, routines to classify characters. This is most easily implemented as a 128-byte table with each byte classifying the corresponding character as special, upper or lower case, numeric, hexadecimal, white space, punctuation, or control character. The routines are then simple masking operations that can now be provided as macros. ■■J


*Part of this work was done during a sabbatical spent at the University of Illinois; in particular, the Small C system was obtained from "UseNet."*
p *can be compiled with the C compiler on Unix/vii.*



**(Listings begin on page 52)**

**P** (Text begins on page 46)

## Listing One

```
/*
 *      p -- small C preprocessor
 *      ats 6/83
 */

/*
 *      define...
 *
 *      verbose         to support -v for debugging
 */

char usage[] =
#ifdef  verbose
        "p [-d n[=v]] [-e] [-i d:] [-u n] [-v] [in [out]]",
        vflag;          /* set by -v */
#else
        "p [-d n[=v]] [-e] [-i d:] [-u n] [in [out]]";
#endif

                        /* command names:                */
#define DEFINE  1       /* # define name text            */
                        /* # define name(name,...) text */
#define ELSE    2       /* # else                        */
#define ENDIF   3       /* # endif                       */
#define IFDEF   4       /* # ifdef name                  */
#define IFNDEF  5       /* # ifndef name                 */
#define INCLUDE 6       /* # include "file"              */
                        /* # include <file>              */
#define LINE    7       /* # line number name            */
#define UNDEF   8       /* # undef name                  */
#define DEFAULT 0       /* any others passed on          */

/*
 *      "FEATURES":
 *
 *              Macro calls must be fully contained on one source
 *              line -- all lines can be continued with \, however.
 *
 *              Recursive definitions are not detected as such.
 *              `p' will report as per NEST.
 *
 *              #else can be used (to reverse the current
 *              #if condition) arbitrarily often.
 */

#include <stdio.h>

/*
 *      i/o header file

#define FILE    ???     type to represent files (used as FILE*)
#define stdin   ???     pre-opened standard input file
#define stdout  ???     pre-opened standard output file
#define stderr  ???     pre-opened diagnostic output file
#define NULL    0       null pointer, false
#define EOF     ???     end of file indication
 */

#include <ctype.h>
```

```
/*
 *      character classification macros header file
 *
 *      isascii(i)      i is ASCII character
 *      isalnum(c)      c is letter or digit
 *      isalpha(c)      c is letter
 *      isdigit(c)      c is digit
 *      islower(c)      c is lower case letter
 *      isspace(c)      c is white space
 *      isupper(c)      c is upper case letter
 *      isxdigit(c)     c is base 16 digit
 *
 *      i can be any integer, isascii(c) must be true for c
 */

#define INCR    80      /* line buffer increment */
#define HASH    128     /* hash table size (power of 2) */
#define NEST    10      /* limit for reprocessing - -1 is "infinite" */

                        /* cmode states */
#define CMcmt   1       /* in comment */
#define CMstr   2       /* in string */
#define CMchr   3       /* in character constant */

#define PARM    0x80    /* flag macro parameter number */
#define PARMNO  0x7f    /* extract parameter number */

/*
 *      special data types
 */

#define LIST        int             /* list of word or string values */
#define l_next(x)   (*(x))          /* -> next element */
#define l_word(x)   ((x)[1])        /* word value */
#define l_str(x)    ((x)+1)         /* -> string value */
#define sz_WORD     4               /* size of word list element */
#define sz_STR(s)   (3+strlen(s))   /* size of string list element */

#define SYMBOL      int             /* list of symbol table elements */
#define s_next(x)   (*(x))          /* -> next element */
#define s_val(x)    ((x)[1])        /* -> defined value */
#define s_name(x)   ((x)+2)         /* -> name */
#define sz_SYM(n)   (5+strlen(n))   /* size of symbol table element */

/*
 *       runtime support routines
 */

extern  _drive(),           /* BDOS function 25: current drive number */
        _narg(),            /* number of arguments passed in this call */
        calloc(),           /* (n,l) return NULL or -> n elements of length l */
        cfree(),            /* (p) free area at p, returned by calloc() */
        exit(),             /* terminate program execution */
        fclose(),           /* (f) close file described by f */
        fgetc(),            /* (f) return EOF or next character from file f */
        fopen(),            /* (n,m) return NULL or descriptor for file "n"
                               opened to read (m == "r"), write ("w"),
                               or append ("a") */
        fputc(),            /* (c,f) write c on file f, return EOF or c */
        fputs(),            /* (s,f) write string s on file f */
        freopen(),          /* (n,m,f) like fopen(), but close and reuse f */
        index(),            /* (s,c) find c in string s, return NULL or -> to it.
                               '\0' is always found */
        itod(),             /* (i) return -> (static) string with i in decimal */
```

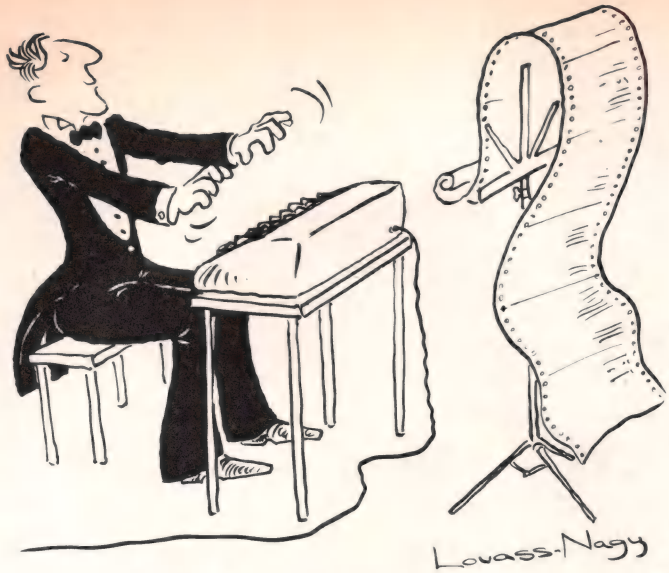
*(Continued on next page)*

P (Listing Continued, text begins on page 46)

## Listing One

```
        strcmp();       /* (a,b) <, ==, > 0 as string a is <, ==, > string b */
        strcpy();       /* (a,b) copy string b to string a */
        strlen();       /* (s) return number of characters in string s */
        strncmp();      /* (a,b,n) like strcmp(), but for n bytes at most */
        strncpy();      /* (a,b,n) like strcpy(), but for n bytes at most */

/*
 *      global data
 */

int     parmno;         /* current number of parameters */
        linelen;        /* current maximum usable length */
        olinelen;
        lineno;         /* current line number */
        olineno;
        iflevel;        /* depth of open #if */
        skip;           /* non-0: iflevel to skip to */

char    eflag;          /* set by -e: prevent position stamps */
        cmode;          /* comment() mode */
        *line;          /* dynamic input line buffer */
        *lp;            /* current position in line */
        *oline;         /* dynamic output line buffer */
        *olp;           /* current position in oline */

LIST    *drive;         /* include prefixes */
        *filenms;       /* open file names */
        *files;         /* open file pointers */
        *lines;         /* line numbers */
        *parms;         /* parameters */

SYMBOL  *symbol;        /* list of symbol table elements */
#ifdef  HASH            /* hash feature (optional) */
                        /* symbol set by find() to -> hashtab at s */
int     hashtab[HASH];  /* really SYMBOL *: begin of chains */
#endif

FILE    *infile;        /* current input file */

main(argc, argv)
        int argc;
        int *argv;
{       char *cp, *vp;

#ifdef  verbose
        LIST *ip;
#endif

        /* current drive is last include prefix */
        vp = "a:";
        *vp += _drive();
        drive = pushs(drive, vp);

        /* predefine "cpm" */
        define("cpm", "");

        /* process arguments, values may be joined or separate */
        while (--argc)
        {       cp = *++argv;
                if (*cp != '-')
```

(Continued on page 56)




# •NEW PRODUCTS•

Before Johann Sebastian Bach developed a new method of tuning, you had to change instruments practically every time you wanted to change keys. Very difficult.

Before Avocet introduced its family of cross-assemblers, developing micro-processor software was much the same. You needed a separate development system for practically every type of processor. Very difficult and very expensive.

But with Avocet's cross-assemblers, a single computer can develop software for virtually any microprocessor! Does that put us in a league with Bach? You decide.

# The Well-Tempered Cross-Assembler

## Development Tools That Work

Avocet cross-assemblers are fast, reliable and user-proven in over 3 years of actual use. Ask NASA, IBM, XEROX or the hundreds of other organizations that use them. Every time you see a new microprocessor-based product, there's a good chance it was developed with Avocet cross-assemblers.

Avocet cross-assemblers are easy to use. They run on any computer with CP/M* and process assembly language for the most popular microprocessor families.

**5¼" disk formats available at no extra cost include Osborne, Xerox, H-P, IBM PC, Kaypro, North Star, Zenith, Televideo, Otrona, DEC.**

## Turn Your Computer Into A Complete Development System

Of course, there's more. Avocet has the tools you need from start to finish to enter, assemble and test your software and finally cast it in EPROM:

**Text Editor VEDIT --** full-screen text editor by CompuView. Makes source code entry a snap. Full-screen text editing, plus TECO-like macro facility for repetitive tasks. Pre-configured for over 40 terminals and personal computers as well as in user-configurable form.

CP/M-80 version ..... $150
CP/M-86 or MDOS version ..... $195
(when ordered with any Avocet product)

**EPROM Programmer** -- Model 7128 EPROM Programmer by GTek programs most EPROMS without the need for personality modules. Self-contained power supply ... accepts ASCII commands and data from any computer through RS 232 serial interface. Cross-assembler hex object files can be down-loaded directly. Commands include verify and read, as well as partial programming.

PROM types supported: 2508, 2758, 2516, 2716, 2532, 2732, 2732A, 27C32, MCM8766, 2564, 2764, 27C64, 27128, 8748, 8741, 8749, 8742, 8751, 8755, plus Seeq and Xicor EEPROMS.

| Avocet Cross-assembler | Target Microprocessor | CP/M-80 Version | •CP/M-86 IBM PC, MSDOS** Versions• |
|---|---|---|---|
| •XASMZ80 | Z-80 | $200.00 each | $250.00 each |
| •XASM85 | 8085 | | |
| XASM05 | 6805 | | |
| XASM09 | 6809 | | |
| XASM18 | 1802 | | |
| XASM48 | 8048/8041 | | |
| XASM51 | 8051 | | |
| XASM65 | 6502 | | |
| XASM68 | 6800/01 | | |
| XASMZ8 | Z8 | | |
| XASMF8 | F8/3870 | | $300.00 each |
| XASM400 | COP400 | | |
| XASM75 | NEC 7500 | $500.00 | |
| Coming soon: XASM68K...68000 | | | |

(Upgrade kits will be available for new PROM types as they are introduced.)

Programmer ..... $389
Options include:
- Software Driver Package -- enhanced features, no installation required.
- CP/M-80 Version ..... $ 75
- IBM PC Version ..... $ 95
- RS 232 Cable ..... $ 30
- 8748 family socket adaptor ... $ 98
- 8751 family socket adaptor ... $174
- 8755 family socket adaptor ... $135

- **G7228 Programmer by GTek** -- baud to 2400 ... superfast, adaptive programming algorithms ... programs 2764 in one minute.
- Programmer ..... $499
- Ask us about Gang and PAL programmers.
- **HEXTRAN Universal HEX File Converter** -- Converts to and from Intel, Motorola, MOS Technology, Mostek, RCA, Fairchild, Tektronix, Texas Instruments and Binary formats.
- Converter, each version ..... $250

## Call Us

If you're thinking about development systems, call us for some straight talk. If we don't have what you need, we'll help you find out who does. If you like, we'll even talk about Bach.

**CALL TOLL FREE 1-800-448-8500**
(In the U.S. except Alaska and Hawaii)

VISA and Mastercard accepted. All popular disc formats now available -- please specify. Prices do not include shipping and handling -- call for exact quotes. OEM INQUIRIES INVITED.

*Trademark of Digital Research **Trademark of Microsoft

**AVOCET SYSTEMS INC.™**
**DEPT. 784 - DDJ**
**804 SOUTH STATE STREET**
**DOVER, DELAWARE 19901**
**302-734-0151 TELEX 467210**

P (Listing Continued, text begins on page 46)

## Listing One

```
                        break;
                switch (cp[1]) {
                case 'd':
                        if (*(cp += 2))
                                ;
                        else if (--argc == 0)
                                goto error;
                        else
                                cp = *++argv;
                        if (vp = index(cp, '='))
                                *vp++ = '\0';
                        else
                                vp = "";
                        define(cp, vp);
                        break;
                case 'e':
                        eflag = 1;
                        break;
                case 'i':
                        /* explicit prefixes in order right to left */
                        if (*(cp += 2))
                                ;
                        else if (--argc == 0)
                                goto error;
                        else
                                cp = *++argv;
                        drive = pushs(drive, cp);
                        break;
                case 'u':
                        if (*(cp += 2))
                                ;
                        else if (--argc == 0)
                                goto error;
                        else
                                cp = *++argv;
                        undefine(cp);
                        break;
#ifdef  verbose
                case 'v':
                        vflag = 1;

                        break;
#endif
                default:
                        goto error;
                }
        }

        /* input file drive is first include prefix */
        vp = "a:";
        *vp += _drive();

        /* allow input and output files */
        switch (argc) {
        case 2:         /* use input, drive(input), output */
        case 1:         /* use input, drive(input) */
                if (cp[1] == ':')
                {       vp = "?:";
                        *vp = cp[0];
                }
```

```
                if (freopen(*argv, "r", stdin) == NULL)
                {       where("cannot read", *argv);
                        exit();
                }
                filenms = pushs(filenms, *argv);
#ifdef  verbose
                if (vflag)
                        where("reading");
#endif
                if (--argc)
                {       if (freopen(*++argv, "w", stdout) == NULL)
                        {       filenms = NULL;
                                where("cannot write", *argv);
                                exit();
                        }
#ifdef  verbose
                        if (vflag)
                                where("writing", *argv);
#endif
                }
        case 0:         /* use stdin, current drive */
                break;
        default:
error:          where(usage);
                exit();
        }

        /* set first include prefix */
        drive = pushs(drive, vp);

#ifdef  verbose
        if (vflag)
        {       for (ip = drive; ip; ip = l_next(ip))
                        where("drive", l_str(ip));
        }
#endif

        /* start reading on stdin */
        infile = stdin;

        /* allocate first buffers */
        if ((line = calloc(INCR, 1)) == NULL
            || (oline = calloc(INCR, 1)) == NULL)
        {       where("no room");
                exit();
        }
        olinelen = linelen = INCR;

        /* make sure, we first get a position stamp */
        olineno = lineno - 3;

        /* main loop */
        while (getline())
                if (! comment() && ! command())
                        process();
}

getline()               /* line = complete line, ascii */
                        /* return false on EOF */
{       int c;          /* current character */

        /* move to lp, concatenating continued lines */
        for (lp = line; ; )
        {       switch (c = fgetc(infile)) {
                case '\\':
```

*(Continued on next page)*

P (Listing Continued, text begins on page 46)

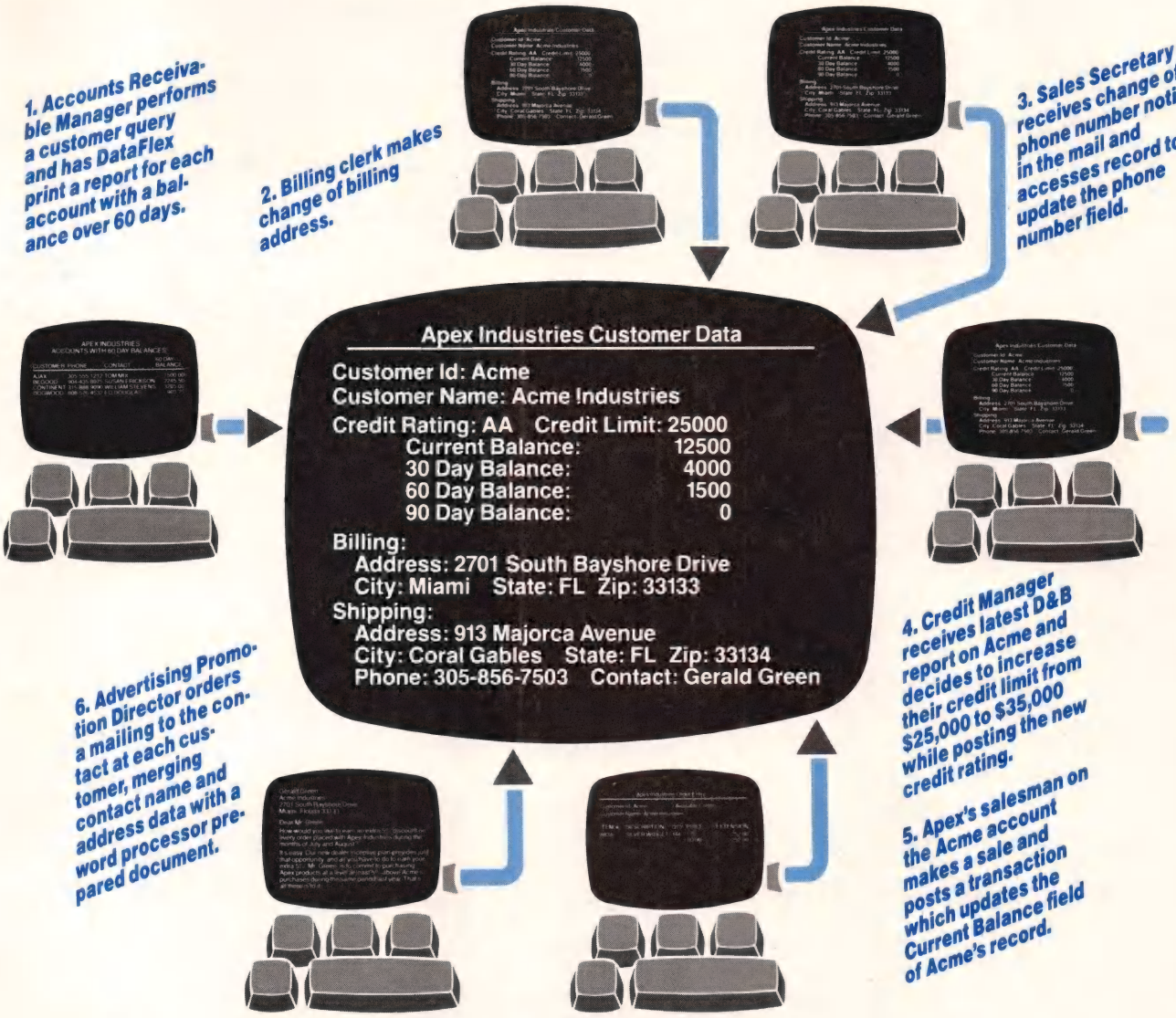
## Listing One

```
                        switch (c = fgetc(infile)) {
                        case EOF:
                                where("trailing \\");
                        case '\n':
                                ++lineno;
                                continue;
                        }
                        in('\\');
                default:
                        if (! isascii(c))
                                where("illegal character");
                        else
                                in(c);
                        continue;
                case EOF:
                        ++ lineno;
                        if (lp != line)
                                break;
                        else if (files)
                        {       fclose(infile);
#ifdef  verbose
                                if (vflag)
                                        where("end include");
#endif
                                infile = pop(&files);
                                lineno = pop(&lines);
                                olineno = lineno - 3; /* stamp! */
                                pop(&filenms);

                                continue;
                        }
                        return 0;
                case '\n':
                        ++lineno;
                        if (lp == line)
                                continue;
                }
                break;  /* got a nonempty line */
        }
        *lp = '\0';
#ifdef  verbose
        if (vflag)
                where("getline", line);
#endif
        return 1;
}

comment()                       /* line = line w/out comments, lead white space */
                                /* return true if comment line */
{       char c;

        /* move from olp to lp, eliminating comments */
        for (lp = olp = line; ; )
        {       switch (c = *olp++) {
                case '\\':
                        if (cmode != CMstr && cmode != CMchr)
                                break;
                        in(c);
                        if (c = *olp++)
                                break;
```

(Continued on page 60,

# ALL AT ONCE!

## AND NEVER A "LOCKED OUT" USER!




DataFlex is the only application development database which ***automatically*** gives you true multi-user capabilities. Other systems can lock you out of records or entire files for the full time they are being used by someone else. DataFlex, however, locks only the data being changed, and ***only*** during the micro-seconds it takes to actually write it to the file! The updated record is then immediately available. The number of users who can access, and change, records at the same time is limited only by the number of terminals on your system or network. Call or write today for all the details on DataFlex...the true multi-user database.

**DATAFLEX**™

DATA ACCESS CORPORATION
8525 SW 129 Terrace, Miami, FL 33156 (305) 238-0012
Telex 469021 DATA ACCESS CI

Compatible with CP/M-80, MSDOS networks, MP/M-86, Novell Sharenet, PC-Net, DMS Hi-net, TurboDOS multi-user, Molecular N-Star, Televideo MmmOST, Action DPC/OS, IBM PC w/Corvus, OMNINET, 3Com EtherSeries and Micromation M/NET.
MSDOS is a trademark of Microsoft. CP/M and MP/M are trademarks of Digital Research.
Circle no. 23 on reader service card.

P (Listing Continued, text begins on page 46)

## Listing One

```
                case '\0':
                        if (cmode == CMstr)
                        {       if (! skip)
                                        where("unbalanced \"");
                                in('\"');
                                cmode = 0;
                        }
                        else if (cmode == CMchr)
                        {       if (! skip)
                                        where("unbalanced \'");
                                in('\'');
                                cmode = 0;
                        }
                        *lp = '\0';
#ifdef  verbose
                        if (vflag)
                                where("comment", line);
#endif
                        return lp == line;
                case '/':
                        if (cmode == 0 && *olp == '*')
                        {       cmode = CMcmt;
                                ++olp;
                                if (lp != line)
                                        in(' ');
                                continue;
                        }
                        break;
                case '*':
                        if (cmode == CMcmt && *olp == '/')
                        {       cmode = 0;
                                ++olp;
                                continue;
                        }
                        break;
                case '\"':
                        switch (cmode) {
                        case 0:
                                cmode = CMstr;
                                break;
                        case CMstr:
                                cmode = 0;
                        }
                        break;
                case '\'':
                        switch (cmode) {
                        case 0:
                                cmode = CMchr;
                                break;
                        case CMchr:
                                cmode = 0;
                        }
                }
                if (cmode != CMcmt && (! isspace(c) || lp != line))
                        in(c);
        }
}

command()               /* process commands */
```

```
                        /* return true if done (i.e., to skip) */
{       int k;
        LIST *dp;
        FILE *fp;
        char *cp;

/*
 *      #if algorithm
 *
 *      skip    if non-zero, knows #if-level to which to skip;
 *              while skipping, comment() is executed, but not process().
 *
 *      iflevel current nesting depth of #if;
 *              counted even while skipping (of course).
 *
 *      #else   if skipping to current #if-level, stop skipping;
 *              if not skipping, start skipping to current level.
 *
 *      In order to limit #else to at most one per #if, we
 *      would need a stack; why bother??
 */
                if (*line != '#' || (k = kind(&lp)) == DEFAULT)
                        return skip;

                /* process the command */
                switch (k) {
                case DEFINE:
                        if (! skip)
                                define(lp, marknm(lp));
                        break;
                case ELSE:
                        if (! skip && iflevel == 0)
                                where("#else without #if");
                        else if (skip == iflevel)
                                skip = 0;
                        else if (skip == 0)
                                skip = iflevel;
                        break;
                case ENDIF:
                        if (! skip && iflevel == 0)
                                where("#endif without #if");
                        else
                        {       if (skip == iflevel)
                                        skip = 0;
                                --iflevel;
                        }
                        break;
                case IFDEF:
                case IFNDEF:
                        ++iflevel;
                        if (! skip)
                        {       marknm(lp);
                                if (isname(lp,""))
                                        if (find(lp))
                                        {       if (k == IFNDEF)
                                                        skip = iflevel;
                                        }
                                        else
                                        {       if (k == IFDEF)
                                                        skip = iflevel;
                                        }
                        }
                        break;
                case INCLUDE:
                        if (! skip)
```

*(Continued on next page)*

**P** (Listing Continued, text begins on page 46)

## Listing One

```
                        if (! markfl(lp))
                                where("#include?");
                        else if (lp[2] == ':')
                        {       ++lp;
                                if (fp = fopen(lp, "r"))
                                {
pushfile:                               files = pushw(files, infile);
                                        lines = pushw(lines, lineno);
                                        lineno = 0;
                                        olineno = lineno - 3; /* stamp ! */
                                        filenms = pushs(filenms, lp);
                                        infile = fp;
#ifdef  verbose
                                        if (vflag)
                                                where("including", lp);
#endif
                                }
                                else
                                        where("cannot open include file", lp);
                        }
                        else
                        {       dp = drive;
                                if (*lp == '(')
                                        dp = l_next(dp);
                                *lp-- = ':';
                                for (; dp; dp = l_next(dp))
                                {       *lp = *l_str(dp);
                                        if (fp = fopen(lp, "r"))
                                                goto pushfile;
                                }
                                where("cannot find include file", lp+2);
                        }
                break;
        case LINE:
                if (! skip)
                {       if (isdigit(*lp))
                        {       for (k = *lp - '0'; isdigit(*++lp); )
                                        k = k*10 + *lp-'0';
                                while (isspace(*lp))
                                        ++lp;
                                cp = lp;
                                while (*lp && ! isspace(*lp))
                                        ++lp;
                                if (lp != cp)
                                {       *lp = '\0';
                                        lineno = k;
                                        if (filenms)
                                                pop(&filenms);
                                        filenms = pushs(filenms, cp);
                                        break;
                                }
                        }
                        where("#line?");
                }
                break;
        case UNDEF:
                if (! skip)
                {       marknm(lp);
                        undefine(lp);
```

```
                }
        }
        return i;
}

process()               /* process regular input line */
{       char expand;    /* reprocess flag */
        char c;         /* current input character */
        SYMBOL *sp;     /* -> found symbol */
        char *name;     /* -> begin of name */
        int i;
        int nest;

        /* expand one buffer into the other */
        nest = NEST+1;
        do
        {       lp = line;
                *(olp = oline) = '\0';
                expand = 0;
                while(c = *lp++)
                        if (isalpha(c) || c == '_')
                        {       name = out(c);
                                while ((c = *lp) && (isalnum(c) || c == '_'))
                                {       out(c);
                                        ++lp;
                                }
                                if (sp = find(name))
                                {       expand = 1;
                                        outmacro(name, sp);
                                }
                        }
                        else if (isdigit(c))
                                lp = outnumber(--lp);
                        else if (c == '\'' || c == 34)
                                lp = outdelim(--lp);
                        else
                                out(c);
                /* if something changed, flip buffers */
                if (expand)
                {       lp = line;
                        line = oline;
                        oline = lp;
                        i = linelen;
                        linelen = olinelen;
                        olinelen = i;
                }
        } while (expand && --nest);
        if (expand)
                where("#define nested too deep");
        output(oline);
}

/*
 *      symbol table routines
 */

define(s, v)            /* #define s v */
        char *s;        /* name of symbol ?? */
        char *v;        /* value */
{       SYMBOL *r;
        int f;
        char *cp, *p, c, *name;

        if (! ismacro(s))
                return;
```

(Continued on page 66)

# Super assemblers plus the world's largest selection of cross assemblers!

**Z-80 Macroassembler $49.50**

**Power for larger programs!** This 2500AD macroassembler includes:

- Zilog Z-80 Macroassembler (with the same powerful features as all our assemblers)
- powerful linker that will link up to 128 files
- Intel 8080 to Zilog Z-80 Source Code Converter (to convert all your Intel source to Zilog Syntax in one simple step)
- COM to Hex Converter (to convert your object files to Hex for PROM creation, etc.)
- 52 pages User Manual

**8086/88 Assembler with Translator $99.50**

**Available for MSDOS, PCDOS, or CPM/86!** This fully relocatable macro-assembler will asemble and link code for MSDOS (PCDOS) AND CPM/86 on either a CPM/86 or MSDOS machine. This package also includes:

- An 8080 to 8086 source code translator (no limit on program size to translate)
- A Z-80 to 8086 translator
- 64 page user manual
- 4 linkers included:
  - –MSDOS produces .EXE file
  - –CPM/86 produces .CMD file
  - –Pure object code generation
  - –Object code and address information only

Linker features:

- Links up to 128 files
- Submit mode invocation
- Code, Data Stack and extra segments
- Handles complex overlays
- Written in assembly language for fast assemblies.
- MICROSOFT .REL format option

**Z-8000 Cross Development Package $199.50**

**Instant Z-8000 Software!** This package allows development and conversion of software for the Z8001, 8002, 8003 and 8004 based machines on a Z-80, Z-8000 or 8086 machine. This powerful package includes:

- a Z-80/8080 to Z-8000 Assembly Language Source Code Translator
- Z-8000 Macro Cross Assembler and Linker

The Translators provide Z-8000 source code from Intel 8080 or Zilog Z-80 source code. The Z-8000 source code used by these packages are the unique 2500AD syntax using Zilog mnemonics, designed to make the transition from Z-80 code writing to Z-8000 easy.

---

***All 2500 AD Assemblers and Cross Assemblers support the following features:***

**Relocatable Code** — the packages include a versatile Linker that will link up to 128 files together, or just be used for external reference resolution. Supports separate Code and Data space. The Linker allows Submit Mode or Command Invocation.

**Large File Handling Capacity** —the Assembler will process files as large as the disk storage device. All buffers including the symbol table buffer overflow to disk.

**Powerful Macro Section**—handles string comparisons during parameter substitutions. Recursion and nesting limited only by the amount of disk storage available.

**Conditional Assembly**—allows up to 248 levels of nesting.

**Assembly Time Calculator**—will perform calculations with up to 16 pending operands, using 16 or 32 Bit arithmetic (32 Bit only for 16 Bit products). The algebraic hierarchy may be changed through the use of parentheses.

**Include files supported—Listing Control**—allows listing of sections on the program with convenient assembly error detection overrides, along with assembly run time commands that may be used to dynamically change the listing mode during assembly.

**Hex File Converter, included**—for those who have special requirements, and need to generate object code in this format.

**Cross reference table generated—**

**Plain English Error Messages—**

System requirements for all programs: Z-80 CP/M 2.2 System with 54k TPA and at least a 96 column printer is recommended. Or 8086/88 256k CP/M-86 or MSDOS (PCDOS).

***Cross Assembler Special Features***

**Z-8**—512 User defined registers names, standard Zilog *and* Z-80 style syntax support.

**8748**—standard Intel *and* Z-80 style syntax supported.

**8051**—512 User defined register or addressable bit names.

**6800 Family**—absolute or relocatable modes, all addressing modes supported, Motorola syntax compatible.

**6502**—Standard syntax or Z-80 type syntax supported, all addressing modes supported.

Circle **no. 1** on reader service card.

## 8086 and Z-8000 XASM includes Source Code Translators

| | Z-80 CP/M® | ZILOG SYSTEM 8000 UNIX | IBM P.C. 8086/88 MSDOS | IBM P.C. 8086/88 CP/M 86 | OLIVETTI M-20 PCOS |
|---|---|---|---|---|---|
| 8086/88 ASM | | | $ 99.50 | $ 99.50 | |
| 8086/88 XASM | $199.50 | $750.00 | | | $199.50 |
| 16000(all) XASM *new* | 199.50 | 750.00 | 199.50 | 199.50 | 199.50 |
| 68000 XASM *new* | 199.50 | 750.00 | 199.50 | 199.50 | 199.50 |
| Z-8000™ ASM | | 750.00 | | | 299.50 |
| Z-8000 XASM | 199.50 | | 199.50 | 199.50 | |
| Z-80 ASM | 49.50 | | | | |
| Z-80 XASM | | 500.00 | 99.50 | 99.50 | 99.50 |
| Z-8 XASM | 99.50 | 500.00 | 99.50 | 99.50 | 99.50 |
| 6301(CMOS) *new* | 99.50 | 500.00 | 99.50 | 99.50 | 99.50 |
| 6500 XASM | 99.50 | 500.00 | 99.50 | 99.50 | 99.50 |
| 6502 XASM | 99.50 | 500.00 | 99.50 | 99.50 | 99.50 |
| 65CO2(CMOS) XASM *new* | 99.50 | 500.00 | 99.50 | 99.50 | 99.50 |
| 6800,2,8 XASM | 99.50 | 500.00 | 99.50 | 99.50 | 99.50 |
| 6801,03 XASM | 99.50 | 500.00 | 99.50 | 99.50 | 99.50 |
| 6805 XASM | 99.50 | 500.00 | 99.50 | 99.50 | 99.50 |
| 6809 XASM | 99.50 | 500.00 | 99.50 | 99.50 | 99.50 |
| 8748 XASM | 99.50 | 500.00 | 99.50 | 99.50 | 99.50 |
| 8051 XASM | 99.50 | 500.00 | 99.50 | 99.50 | 99.50 |
| 8080 XASM | 99.50 | 500.00 | 99.50 | 99.50 | 99.50 |
| 8085 XASM *new* | 99.50 | 500.00 | 99.50 | 99.50 | 99.50 |
| 1802 XASM *new* | 99.50 | 500.00 | 99.50 | 99.50 | 99.50 |
| F8/3870 XASM *new* | 99.50 | 500.00 | 99.50 | 99.50 | 99.50 |
| COPS400 XASM *new* | 99.50 | 500.00 | 99.50 | 99.50 | 99.50 |
| NEC7500 XASM *new* | 99.50 | 500.00 | 99.50 | 99.50 | 99.50 |
| NSC800 *new* | 99.50 | 500.00 | 99.50 | 99.50 | 99.50 |
| **Subtotal** | $ _____ | $ _____ | $ _____ | $ _____ | $ _____ |

Name _____

Company _____

Address _____

City _____ State _____ Zip _____

Phone _____ Ext. _____

Make and model of computer system _____

☐ C.O.D. *(2500AD pays C.O.D. charges)*

☐ VISA or MasterCard #, Exp. Date (mo./yr.)

_____

***TO ORDER.*** **Simply circle the product or products you want in the price columns above, enter the subtotal at the bottom of that column and add up your total order. Don't forget shipping/handling.**

**Total $** _____

**Check one:**

☐ 8″ Single Density
☐ 5¼″ Osborne
☐ IBM P.C.
☐ Cartridge Tape
☐ Apple (Softcard)

shipping/handling ($6.50 per unit, $20.00 per unit for Int'l. airmail) $ _____

**Total Order $** _____

CP/M is a registered trademark of Digital Research, Inc.

Signature _____

# 2500AD SOFTWARE INC.

P.O. Box 441410, Aurora, CO 80014, 303-752-4382 TELEX 752659/AD

P (Listing Continued, text begins on page 46)

## Listing One

```
        /* prune and parametrize value */
        while (isspace(*v))
                ++v;
        if (*v)
        {       for (cp = v + strlen(v); cp != v; )
                        if (! isspace(*--cp))
                                break;
                        else
                                *cp = '\0';
                /* if we have parameters, replace names by positions */
                if (parmno)
                {       p = cp = v;
                        while (c = *cp++)
                                if (isalpha(c) || c == '_')
                                {       *(name = p++) = c;
                                        while ((c = *cp)
                                                && (isalnum(c) || c == '_'))
                                        {       *p++ = c;
                                                ++cp;
                                        }
                                        if (f = findparm(name, p-name))
                                        {       *(p = name) = f | PARM;
                                                ++p;
                                        }
                                }
                                else
                                {       *p++ = c;
                                        /* name as trailer of a constant?? */
                                        if (c == '0'
                                                && (*cp == 'x' || *cp == 'X'))
                                                do
                                                        *p++ = *cp++;
                                                while (isxdigit(*cp));
                                        else if (isdigit(c))
                                                while (isdigit(*cp))
                                                        *p++ = *cp++;
                                        else if (c == '\\')
                                                if (*cp)
                                                        *p++ = *cp++;
                                }
                        *p = '\0';
                }
        }

        /* check if (different) redefinition */
        if (r = find(s))
        {       if (strcmp(s_val(r), v) != 0)
                        where("redefining", s);
                undefine(s);    /* parmno may change */
#ifdef  verbose
                if (vflag)
                        fputs("redefine ", stderr);
#endif
        }
        /* if parametrized, save count */
        if (parmno)
        {       cp = s + strlen(s);
                *cp = '(';
                *++cp = parmno;
```

(Continued on page 68)

# THE PROGRAMMER'S SHOP

helps compare, evaluate, find products. Straight answers for serious programmers.

## SERVICES

- Programmer's Referral List
- Compare Products
- Help find a Publisher
- Evaluation Literature free
- BULLETIN BOARD - 7 PM to 7 AM 617-461-0174
- Dealer's Inquire
- Newsletter
- Rush Order
- Over 300 products

## Our Free Report: PRODUCTIVITY - MSDOS

Assume use of compiler and typical editor. What commercial or public domain products, what techniques improve productivity? "Productivity with MSDOS" is a growing document with some answers. Call to request it. **Help improve it. Earn $50** credit toward any purchase when we add any description, code, or idea received from you.

## RECENT DISCOVERIES

PROFILER - Examine MSDOS program execution speeds. Determine where to improve programs in any Microsoft language, Lattice, or C86. Make histograms that show time spent in portions of your program, and doing MSDOS I/O, etc. $175.

## "C" LANGUAGE

| | | LIST PRICE | OUR PRICE |
|---|---|---|---|
| APPLE: AZTEC C - Full, ASM | | $199 | call |
| 8080: BDS C - Fast, popular | | 150 | 125 |
| 8080: AZTEC C - Full | | 199 | call |
| Z80: ECOSOFT - Fast, Full | | 250 | 225 |
| 8086: C86 - optimizer, Meg | | 395 | call |
| 8086: Lattice - New 1.1 & 2.0 | | 500 | call |
| Microsoft (Lattice) | MSDOS | 500 | call |
| Digital Research - Megabyte | 8086 | 350 | 269 |
| Desmet by CWare - Fast | 8086 | 109 | 99 |

## BASIC

| | ENVIRONMENT | | |
|---|---|---|---|
| Active Trace - debug | 8080/86 | $ 80 | 72 |
| MBASIC-80 - MicroSoft | 8080 | 375 | 255 |
| BASCOM-86 - MicroSoft | 8085 | 395 | 279 |
| CB-86 - DRI | CPM86 | 600 | 439 |
| Prof. BASIC Compiler | PCDOS | 345 | 325 |
| BASIC Dev't System | PCDOS | 79 | 72 |

## EDITORS Programming

| | | | |
|---|---|---|---|
| C Screen with source | 8080/86 | NA | 60 |
| EDIX - clean | PCDOS | 195 | 149 |
| FINAL WORD - for manuals | 8080/86 | 300 | 215 |
| MINCE - like EMACS | CPM, PCDOS | 175 | 149 |
| PMATE - powerful | CPM | 195 | 175 |
| | 8086 | 225 | 195 |
| VEDIT - full, liked | CPM, PCDOS | 150 | 119 |
| | 8086 | 200 | 159 |

## FORTRAN

| | | | |
|---|---|---|---|
| MS FORTRAN-86 - Meg | MSDOS | $350 | $255 |
| SS FORTRAN - 86 | CPM-86 | 425 | 345 |
| FORTRAN-80 - 66 decent | CPM-80 | 500 | 350 |
| INTEL FORTRAN - 86 | IBM PC | NA | 1400 |
| DR FORTRAN COMING | | | |
| RM FORTRAN COMING | | | |

## LANGUAGE LIBRARIES

| | | | |
|---|---|---|---|
| C to dBASE interface | 8080/85 | $125 | $115 |
| C Tools 1 - String, Screen | PCDOS | NA | 115 |
| C Tools 2 - OS Interface | PCDOS | NA | 92 |
| FLOAT 87 - Lattice, PL1 | PCDOS | NA | 115 |
| GRAPHICS: GSX - 80 | CPM80 | NA | 75 |
| HALO - fast, full | PCDOS | 200 | 175 |
| Greenleaf for C - full, 200+ | PCDOS | NA | 165 |
| ISAM: Access Manager - 86 | 8086 | 400 | 300 |
| BTRIEVE - many languages | PCDOS | 245 | 215 |
| PHACT - with C | PCDOS | NA | 250 |
| FABS | CPM80 | 150 | 135 |
| PASCAL TOOLS - Blaise | PCDOS | NA | 115 |
| SCREEN: Display Mgr. 86 | 8086 | 500 | 375 |
| PANEL-86 - many languages | PCDOS | 295 | 245 |
| **WINDOWS for C** | PCDOS | NA | 115 |
| Virtual Screen - Amber | PCDOS | 295 | call |

## PASCAL

| | ENVIRONMENT | LIST PRICE | OUR PRICE |
|---|---|---|---|
| PASCAL MT + 86 | CPM86/IBM | $400 | $289 |
| without SPP | CPM80 | 350 | 249 |
| MS PASCAL 86 | MSDOS | 350 | 255 |
| SBB PASCAL-great, fast | PCDOS | 350 | 315 |
| PASCAL 64 - nearly full | COM 64 | 99 | 89 |
| **SBB Jr - best to learn** | PCDOS | NA | 95 |

## OTHER PRODUCTS

| | | | |
|---|---|---|---|
| Assembler & Tools - DRI | 8086 | 200 | 159 |
| COBOL - Level II | 8086 | 1600 | 1275 |
| CODESMITH-86 - debug | PCDOS | 149 | 139 |
| GC LISP | PCDOS | 495 | 475 |
| IQ LISP - full 1000K RAM | PCDOS | 175 | call |
| Janus ADA - solid value | PCDOS | 500 | 449 |
| **MBP Cobol-86 - fast** | 8086 | 750 | 695 |
| Microshell improve CPM | 8080 | 150 | 125 |
| Microsoft MASM-86 | MSDOS | 100 | 85 |
| PL/1-86 | 8086 | 750 | 560 |
| PLINK-86 - overlays | 8086 | 350 | 315 |
| POWER - recover files | 8080/86 | 169 | 139 |
| READ CPM86 from PCDOS | PCDOS | NA | 55 |
| READ PCDOS on an IBM PC | CPM86 | NA | 55 |
| Trace 86 | PCDOS | 125 | 115 |

## FEATURES

C INTERPRETERS for MSDOS - Ask about one for beginners for $85 or full development for $500.

C HELPER includes source in C for MSDOS or CPM80 for a DIFF, GREP, Flowcharter, C Beautifier and others. Manage your source code easier. $125.

PROLOG86 Interpreter for MSDOS includes tutorials, reference and good examples. Learn in first few hours. For Prototyping, Natural Language or AI. $125.

Call for a catalog, literature, and answers

**800-421-8006**

**THE PROGRAMMER'S SHOP™**

128-D Rockland Street, Hanover, MA 02339.

Visa 617-826-7531, Mass: 800-442-8070 MasterCard

Note: All prices subject to change without notice. Mention this ad. Some prices are specials.

Ask about COD and POs.
Special formats available.

Circle **no. 51** on reader service card.

# PROLOG-86™

## Learn Fast, Experiment, Prototype

1 or 2 pages of PROLOG would require 10 or 15 pages in "C" - and PROLOG is much easier.

In **one evening** develop a feel for PROLOG. In a **few days** understand and enhance artificial intelligence programs included like:

*** an Expert System**
*** a Natural Language Processor**

PROLOG-86 includes two tutorials, a reference, 6 sample programs and a PROLOG Interpreter.

Intro price: $125 for PCDOS, MSDOS or CPM-86. Call:

**Full Refund** if not satisfied in first 3 weeks.

**SOLUTION SYSTEMS**
45-D Accord Park Drive, Norwell, MA 02061
**617-871-5435**

Circle **no. 52** on reader service card.

# C Helper™

## UNIX-like Utilities for C Programming with source

**Save time** when working with your C programs. Full source lets you make them work your way, helps you learn.

Utilities included: compare files (DIFF), cross reference your variables (CCREF), examine the flow of functions as they call each other (FCHART), format and indent programs (pretty printer), search for patterns (GREP). Others check program syntax, print programs your way and more.

$135 for PCDOS, MSDOS, CPM-86 or CPM80.

Call with questions or for "Programming with C Helper".

**SOLUTION SYSTEMS**
45-D Accord Park Drive, Norwell, MA 02061
**617-871-5435**

Circle **no. 53** on reader service card.

(Listing Continued, text begins on page 46)

## Listing One

```
                *++cp = '\0';
        }


        /* ready to make new entry */
        if ((r = calloc(sz_SYM(s), 1)) == NULL)
        {       where("no room");
                exit();
        }
        else
        {
#ifdef  HASH    /* find() sets symbol -> hashtab at s */
                s_next(r) = *symbol;
                *symbol = r;
#else
                s_next(r) = symbol;
                symbol = r;
#endif
                s_val(r) = NULL;
                strcpy(s_name(r), s);
#ifdef  verbose
                if (vflag)
                        fputs("define ", stderr);
#endif
        }


        /* save new value */
        if ((s_val(r) = calloc(strlen(v)+1, 1)) == NULL)
        {       where("no room");
                exit();
        }
        strcpy(s_val(r), v);
#ifdef  verbose
        if (vflag)
        {       fputs(s, stderr);
                fputc(' ', stderr);
                fputs(s_val(r), stderr);
                fputc('\n', stderr);
        }
#endif
}

undefine(s)                     /* #undef s */
        char *s;                /* name of symbol ?? */
{       SYMBOL *r, *p;

        if (isname(s, "") && (r = find(s)))
        {       cfree(s_val(r));
                /* need to unlink symbol descriptor from chain */
#ifdef  HASH    /* find() sets symbol -> hashtab at s */
                if (r == *symbol)
                        *symbol = s_next(*symbol);
                else
                {       for (p = *symbol;
#else
                if (r == symbol)
                        symbol = s_next(symbol);
                else
```

```
                {       for (p = symbol;
#endif
                                        s_next(p) != r; p = s_next(p))
                                ;
                        s_next(p) = s_next(r);
                }
                cfree(r);
#ifdef  verbose
                if (vflag)
                        where("undefine", s);
#endif
        }
}

find(s)                 /* locate s in symbol table */
                        /* return NULL or -> entry */
        char *s;        /* name to find */
{       SYMBOL *r;
        char *sp, *rp, c;
#ifdef  HASH
        int h;

        /* symbol table chains start in hashtab[] */
        /* compute hash address as sum of letters */
        for (h = 0, sp = s; c = *sp; ++sp)
                h += c;
        symbol = hashtab + (h & (HASH-1));

        /* run down the chain */
        for (r = *symbol;
#else
        /* symbol table chain is one linear list */
        /* run down the chain */
        for (r = symbol;
#endif
                        r; r = s_next(r))
        {       for (sp = s, rp = s_name(r); (c = *sp) && *rp == c; ++sp, ++rp)
                        ;
                if (c == '\0' && (*rp == '\0' || *rp == '('))
                        return r;
        }
        return NULL;
}

findparm(s, l)          /* return 0 or parameter number */
        char *s;        /* -> begin of possible parameter name */
        int l;          /* length of name */
{       int f;
        LIST *p;

        for (f = 0, p = parms; p; ++f, p = l_next(p))
                if (strncmp(l_str(p), s, l) == 0)
                        return parmno - f;
        return 0;
}

isname(s,d)             /* true, if s is a name */
                        /* return -> delimeter or NULL */
        char *s;        /* -) begin of name */
        char *d;        /* chars in which name may also end */
{       char *cp, c;

        for (cp = s; index(d, c = *cp) == NULL; ++cp)
                if (! isalnum(c) && c != '_')
                        goto error;
```

*(Continued on next page)*

P (Listing Continued, text begins on page 46)

## Listing One

```
        if (cp == s || isdigit(*s))
        {
error:          where("illegal name", s);
                return NULL;
        }
        return cp;      /* return -> delimeter */
}

ismacro(s)              /* true, if s is a macro header */
        char *s;        /* -> begin of name or header */
{       char *cp, c;

        while (parms)   /* free old parameter list */
                pop(&parms);
        parmno = 0;
        if ((s = isname(s, "(")) == NULL)
                return 0;
        if (*s)         /* we have a new macro */
        {       *s = '\0';      /* delimit name */
                do              /* and parse parameters */
                {       while (isspace(*++s))
                                ;
                        if (cp = isname(s, ",) \t"))
                        {       c = *cp;
                                *cp = '\0';
                                parms = pushs(parms, s);
                                ++ parmno;
                        }
                        else
                                return 0;
                        while (isspace(c))
                                c = *++cp;
                        s = cp;
                } while (c == ',');
                if (c != ')')
                {       where("illegal macro header");
                        return 0;
                }
        }
        return 1;
}

markmm(s)               /* bypass and terminate macro header */
                        /* return -> value */
        char *s;        /* ->begin of name */
{       char c;

        /* find white space or ( */
        while ((c = *s) && ! isspace(c) && c != '(')
                ++s;

        /* if (, there must be names, white space and then ) */
        if (c == '(')
        {       while ((c = *++s) && c != ')')
                        ;
                /* after ) there must be \0 or white space */
                if (c && (s[1] == '\0' || isspace(s[1])))
                        ++s;
        }
```

(Continued on page 72)

# TOTAL CONTROL:

## *FORTH:* FOR Z-80®, 8086, 68000, and IBM® PC

**Complies with the New 83-Standard**

**GRAPHICS • GAMES • COMMUNICATIONS • ROBOTICS**
**DATA ACQUISITION • PROCESS CONTROL**

- ***FORTH*** programs are instantly portable across the four most popular microprocessors.
- ***FORTH*** is interactive and conversational, but 20 times faster than BASIC.
- ***FORTH*** programs are highly structured, modular, easy to maintain.
- ***FORTH*** affords direct control over all interrupts, memory locations, and i/o ports.
- ***FORTH*** allows full access to DOS files and functions.
- ***FORTH*** application programs can be compiled into turnkey COM files and distributed with no license fee.
- ***FORTH*** Cross Compilers are available for ROM'ed or disk based applications on most microprocessors.

Trademarks: IBM, International Business Machines Corp.; CP/M, Digital Research Inc.; PC/Forth + and PC/GEN, Laboratory Microsystems, Inc.

**FORTH Application Development Systems** include interpreter/compiler with virtual memory management and multi-tasking, assembler, full screen editor, decompiler, utilities and 200 page manual. Standard random access files used for screen storage, extensions provided for access to all operating system functions.

**Z-80 FORTH** for CP/M® 2.2 or MP/M II, $100.00; **8080 FORTH** for CP/M 2.2 or MP/M II, $100.00; **8086 FORTH** for CP/M-86 or MS-DOS, $100.00; **PC/FORTH** for PC-DOS, CP/M-86, or CCPM, $100.00; **68000 FORTH** for CP/M-68K, $250.00.

**FORTH + Systems** are 32 bit implementations that allow creation of programs as large as 1 megabyte. The entire memory address space of the 68000 or 8086/88 is supported directly.

| | |
|---|---|
| **PC FORTH +** | $250.00 |
| **8086 FORTH +** for CP/M-86 or MS-DOS | $250.00 |
| **68000 FORTH +** for CP/M-68K | $400.00 |

**Extension Packages** available include: software floating point, cross compilers, INTEL 8087 support, AMD 9511 support, advanced color graphics, custom character sets, symbolic debugger, telecommunications, cross reference utility, B-tree file manager. *Write for brochure.*

VISA

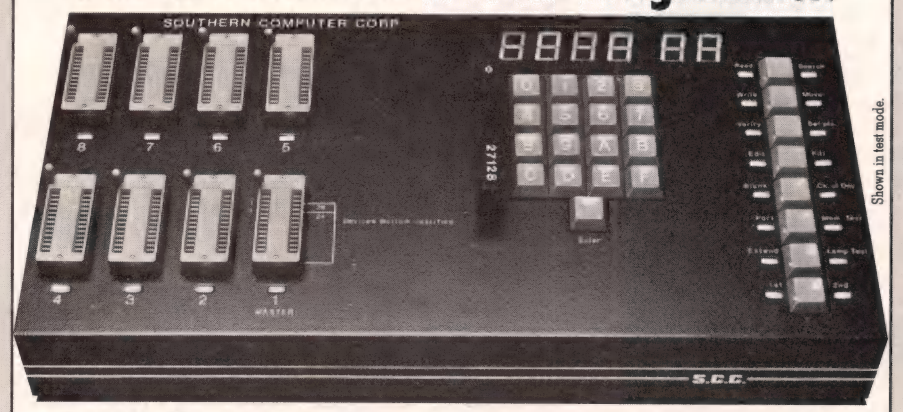
LMI

***Laboratory Microsystems Incorporated***
***Post Office Box 10430, Marina del Rey, CA 90295***
*Phone credit card orders to* ***(213) 306-7412***

MasterCard

Circle **no. 35** on reader service card.

# The Cost Efficient EPROM Programmer!




Shown in test mode.

**DISPLAY** ☐ Bright 1" high display system ☐ Progress indicated during programming ☐ Error messages

**KEYBOARD** ☐ Full travel entry keys ☐ Auto repeat ☐ Illuminated function indicators

**INTERFACE** ☐ RS-232C for data transfer ☐ 110-19.2K baud ☐ X-on X-off control of serial data

**FUNCTIONS** ☐ Fast and standard programming algorithms ☐ Single key commands ☐ Search finds data strings up to 256 bytes long ☐ Electronic signatures for easy data error I.D. ☐ "FF" skipping for max programming speed ☐ User sets memory boundaries ☐ 15 commands including move, edit, fill, search, etc. functions ☐ Extended mode reads EPROM sets

**GENERAL** ☐ Stand alone operation, *external terminal not needed for full command set* ☐ Total support ☐ 28 pin sockets ☐ Faulty EPROMS indicated at socket ☐ Programs 1 to 128K devices ☐ Built in diagnostics ☐ No calibration required ☐ No personality modules to buy ☐ Complete with 128K buffer ☐ Only

**$995.00 COMPLETE**

Dealer inquiries welcome.

**SOUTHERN COMPUTER CORPORATION**
**3720 N. Stratford Rd., Atlanta, GA 30342, 404-231-5363**

Circle **no. 70** on reader service card.

# DeSmet C

## The fastest 8088 C Compiler available

**FULL DEVELOPMENT PACKAGE**
- C Compiler
- Assembler
- Linker and Librarian
- Full-Screen Editor
- Newsletter for bugs/updates

**SYMBOLIC DEBUGGER**
- Monitor and change variables by name using C expressions
- Multi-Screen support for debugging PC graphics and interactive systems
- Optionally display C source during execution
- Breakpoint by Function and Line #

**COMPLETE IMPLEMENTATION**
- **Both** 1.0 **and** 2.0 DOS support
- Everything in K&R (incl. STDIO)
- Intel assembler mnemonics
- **Both** 8087 **and** Software Floating Point

**OUTSTANDING PERFORMANCE**

**Sieve** Benchmark

| | |
|---|---|
| COMPILE | 4 Sec. RAM — 22 Sec FDISK |
| LINK | 6 Sec. RAM — 34 Sec. FDISK |
| RUN | 12 Sec. |
| SIZE | 8192 bytes |

**DeSmet C Development Package $159**

To Order Specify:

Machine _____

OS ☐ MS-DOS ☐ CP/M-86

Disk ☐ 8" ☐ 5¼ SS ☐ 5¼ DS

CWARE CORPORATION

**P.O. BOX 710097**
**San Jose, CA 95171-0097**
**(408) 736-6905**

California residents add sales tax. Shipping: U.S. no charge, Canada add $5, elsewhere add $15. Checks must be on a US Bank and in US Dollars.

Circle **no. 22** on reader service card.

**P** (Listing Continued, text begins on page 46)

## Listing One

```
        /* terminate in place of white space */
        if (*s)
                *s++ = '\0';

        /* this is a rough draft -- see ismacro/isname */
        return s;
}

/*
 *      input and output routines
 */

in(c)                   /* store incoming character */
        char c;         /* to be stored at lp */
{
        *lp++ = c;
        if (lp >= line+linelen)
                rebuff(&lp, &line, &linelen);
}

out(c)                  /* store a character, return -> stored char */
        char c;         /* to be stored at olp */
{
        *olp++ = c;
        if (olp >= oline+olinelen)
                rebuff(&olp, &oline, &olinelen);
        *olp = '\0';    /* maintain trailer */
        return olp-1;
}

rebuff(p, buf, len)     /* make buffer longer */
        int *p;         /* & current pointer */
        int *buf;       /* & buffer pointer */
        int *len;       /* & maximum length */
{
        if ((*p = calloc(*len + INCR, 1)) == NULL)
        {       where("no room");
                exit();
        }
        strncpy(*p, *buf, *len);
        cfree(*buf);
        *buf = *p;
        *p = *buf + *len;
        *len += INCR;
}

output(s)               /* write a string */
        char *s;        /* to write as a line */
{
        /* synchronize output linecount */
        if (! eflag && ++olineno != lineno)
        {       if (++olineno != lineno)
                {       fputc('#', stdout);
                        fputs(itod(olineno = lineno), stdout);
                        if (filenms)
                        {       fputc(' ', stdout);
                                fputs(l_str(filenms), stdout);
                        }
                }
```

```
                fputc('\n', stdout);
        }

        /* emit string as a line */
        fputs(s, stdout);
        if (fputc('\n', stdout) == EOF)
        {       where("output file full");
                exit();
        }
}

/*
 *      C constant processing:
 *
 *      digits          decimal
 *      0digits         octal
 *      0xdigits        hexadecimal
 *      'c'             character value (escapes ok)
 */

outnumber(cp)           /* store a C constant in decimal */
                        /* return -> past it */
        char *cp;       /* -> constant text (digit) */
{       char c, *p;
        int base;
        int i;

        base = 10;
        i = 0;
        if ((c = *cp) == '0')
        {       base = 010;
                if ((c = *++cp) == 'x' || c == 'X')
                {       base = 0x10;
                        c = *++cp;
                }
        }
        for ( ; c; c = *++cp)
        {       if (isdigit(c))
                        c -= '0';
                else if (isxdigit(c))
                        if (isupper(c))
                                c -= 'A' - 10;
                        else
                                c -= 'a' - 10;
                else
                        break;
                if (c < base)
                        i = i*base + c;
                else
                        break;
        }
        for (p = itod(i); c = *p; ++p)
                out(c);
        return cp;
}

outdelim(cp)            /* store a delimited string, return -> past trailer */
        char *cp;       /* -> delimeter */
{       char c, *p;

        if ((c = *cp) == '\"')
        {       out(c);
                while (c = *++cp)
                {       out(c);
                        if (c == '\"')
```

*(Continued on next page)*

P (Listing Continued, text begins on page 46)

## Listing One

```
                        return cp+1;
                if (c == '\\')
                        if (c = *++cp)
                                out(c);
                        else
                                break;
        }
    }
    else    /* it must be character constant */
        switch (c = *++cp) {
        case 0:
        case '\'':
                goto error;
        case '\\':
                switch (c = *++cp) {
                case 'b':  c = '\b'; break;
                case 'f':  c = '\f'; break;
                case 'n':  c = '\n'; break;
                case 'r':  c = '\r'; break;
                case 't':  c = '\t';
                case '\'':
                case '\\':
                case '\"':          break;
                default:
                        if (! isdigit(c) || (c -= '0') > 7)
                                goto error;
                        if (isdigit(cp[1]) && cp[1] <= '7')
                        {       c = (c << 3) + *++cp - '0';
                                if (isdigit(cp[1]) && cp[1] <= '7')
                                        c = (c << 3) + *++cp - '0';
                        }
                }
        default:
                if (*++cp != '\'')
                {
error:                  where("illegal character constant");
                        while (*cp && *cp != '\'')
                                ++cp;
                        if (*cp)
                                ++cp;
                        break;
                }
                for (p = itod(c); *p; ++p)
                        out(*p);
                out(' ');
                ++cp;
        }
    return cp;
}

/*
 *      macro processing
 */

outmacro(at, s)         /* replace string by macro value */
    char *at;           /* replace from here on */
    SYMBOL *s;          /* using this definition */
{   char *vp, c;
```

(Continued on page 76)

Circle **no. 8** on reader service card.



Circle **no. 40** on reader service card.



Circle **no. 3** on reader service card.

**P** (Listing Continued, text begins on page 46)

## Listing One

```
        /* set output up for replacement */
        olp = at;

        /* force white space around replacement */
        if (olp ) oline && (! isspace(olp[-1])))
                out(' ');

        /* if parametrized, collect arguments */
        if (vp = index(s_name(s), '('))
                markarg(++vp);
        else
                parmno = 0;

        /* emit replacement */
        for (vp = s_val(s); c = *vp++; )
                if (c & PARM)
                        outarg(c & PARMNO);
                else
                        out(c);
        /* white space */
        out(' ');
}

markarg(n)              /* mark and collect arguments */
        int n;          /* number to find */
{       char c, cmode;  /* cmode during argument collection only */
        int lpar;

        /* release parameter/argument list, if any */
        while (parms)
                pop(&parms);
        parmno = 0;

        /* find ( */
        while (isspace(c = *lp))
                ++lp;

        /* collect */
        if (c == '(')
        {       do
                {       parms = pushw(parms, ++lp);
                        ++ parmno;
                        lpar = cmode = 0;
                        for ( ; c = *lp; ++lp)
                        {       switch (c) {
                                case '(':
                                        if (cmode == 0)
                                                ++ lpar;
                                        continue;
                                case ',':
                                        if (cmode || lpar)
                                                continue;
                                        break;
                                case ')':
                                        if (cmode || lpar --)
                                                continue;
                                        break;
                                case '\'':
                                        switch (cmode) {
```

```
                                        case 0:
                                                cmode = CMchr;
                                        case CMstr:
                                                continue;
                                        }
                                        cmode = 0;
                                        continue;
                                case '\"':
                                        switch (cmode) {
                                        case 0:
                                                cmode = CMstr;
                                        case CMchr:
                                                continue;
                                        }
                                        cmode = 0;
                                        continue;
                                case '\\':
                                        if (*++lp == '\0')
                                                break;
                                default:
                                        continue;
                                }
                                *lp = '\0';
                                break;
                        }
                } while (c == ',');
                if (c == ')')
                        ++lp;
                else
                        where("incomplete macro call");
        }

        /* check and fill argument count */
        if (parmno != n)
                where("wrong number of arguments");
        for ( ; parmno < n; ++parmno)
                parms = pushw(parms, "");
}

outarg(i)               /* emit argument */
        int i;          /* number to emit */
{       LIST *p;
        char *cp, c;

        /* play double safe */
        if (i > parmno)
        {       where("outarg()))??");
                exit();
        }

        /* locate */
        for (i = parmno-i, p = parms; i && p; --i, p = l_next(p))
                ;
        if (p == NULL)
        {       where("outarg(NULL)??");
                exit();
        }

        /* emit, no white space */
        for (cp = l_word(p); c = *cp; ++cp)
                out(c);
}

/*
 *      stack routines
```

*(Continued on next page)*

P (Listing Continued, text begins on page 46)

## Listing One

```
 */

pushw(l, w)                 /* push word, return -> new list */
        LIST *l;            /* list */
        int w;              /* word to push */
{       LIST *r;
        if ((r = calloc(sz_WORD, 1)) == NULL)
        {       where("no room");
                exit();
        }
        l_next(r) = l;
        l_word(r) = w;
        return r;
}

pushs(l, s)                 /* push string, return -> new list */
        LIST *l;            /* list */
        char *s;            /* string to push */
{       LIST *r;

        if ((r = calloc(sz_STR(s), 1)) == NULL)
        {       where("no room");
                exit();
        }
        l_next(r) = l;
        strcpy(l_str(r), s);
        return r;
}

pop(l)                      /* pop list, return word */
        LIST *l;            /* really **: list header */
{       LIST *r;
        int i;

        if (*l == NULL)
        {       where("pop(NULL)??");
                exit();
        }
        r = *l;             /* element to pop */
        i = l_word(r);      /* result */
        r = l_next(r);      /* following element */
        cfree(*l);
        *l = r;
        return i;           /* nonsense for a string list */
}

/*
 *      other utilities
 */

where(vararg)               /* error message writer */
        int vararg;         /* arbitrarily many strings */
{       int narg, *argv;

        narg = _narg();
        argv = &vararg;
        argv += narg;
        if (filenms)
```

(Continued on page 80)

# SMALL C FOR IBM-PC

Small-C Compiler Version 2.1 for PC-DOS/MS-DOS
Source Code included for Compiler & Library
New 8086 optimizations
Rich I/O & Standard Library

**$40**

## CBUG SOURCE LEVEL DEBUGGER FOR SMALL C

Break, Trace, and Change variables all on the source level
Source code included

**$40**

**Datalight**
11557 8th Ave. N.E.
Seattle, Washington 98125
**(206) 367-1803**

ASM or MASM is required with compiler.
Include disk size (160k/320k), and DOS version with order.
VISA & MasterCard accepted. Include card no. & expiration date.
Washington state residents include 7.9% sales tax.
IBM-PC & PC-DOS are trademarks of International Business Machines
MS-DOS is a trademark of Microsoft Corporation.

Circle **no. 24** on reader service card.

**LEO ELECTRONICS, INC.**
2730 Monterey Street, Suite 111
Torrance, California 90503
**(213) 212-6133 • (800) 421-9565**
**Telex: 291 986 LEO UR**

## PRICE ! QUALITY ! SERVICE !

### RAMS

| | | |
|---|---|---|
| 4116 (150ns) | 1.35 ) | 16K UPGRADE |
| 4116 (200ns) | 1.25 ) | |
| 4164 (150ns) | 4.95 ) | 64K UPGRADE |
| 4164 (200ns) | 4.75 ) | |
| 6116P-3 | 4.40 | |

### EPROMS

| | | | |
|---|---|---|---|
| 2708 | 3.00 | 2532 | 4.50 |
| 2716 | 3.20 | 2732 | 3.95 |
| TMS2716 | 4.75 | 2764 | 7.00 |

**TERMS:** Check, Visa, Mastercard. Call for C.O.D.
U.S. Funds only. California residents add 6½% sales tax.
**SHIPPING:** Add $2.00 for Ground and $5.00 for Air.
**ALL MAJOR MANUFACTURERS**
**ALL PARTS 100% GUARANTEED**
Pricing subject to change without notice.

Circle **no. 37** on reader service card.

# GTEK INC.

DEVELOPMENT HARDWARE/SOFTWARE
HIGH PERFORMANCE/ COST RATIO
(601) 467-8048

## EPROM PROGRAMMER

Compatible w/all Rs 232 serial interface port • Auto select baud rate • With or without handshaking • Bidirectional Xon/Xoff and CTS/DTR supported • Read pin compatible ROMS • No personality modules • Intel, Motorola, MCS86, Hex formats • Split facility for 16 bit data paths • Read, program, formatted list commands • Interrupt driven. program and verify real time while sending data • Program single byte, block, or whole EPROM • Intelligent diagnostics discern bad and erasable EPROM • Verify erasure and compare commands • Busy light • Complete w/Textool zero insertion force socket and integral 120 VAC power (240 VAC/50Hz available)

DR Utility Package allows communication with 7128, 7228, and 7956 programmers from the CP/M command line. Source Code is provided. PGX utility package allows the same thing, but will also allow you to specify a range of addresses to send to the programmer, Verify, set the Eprom type.

**MODEL 7316 PAL PROGRAMMER**
Programs all series 20 PALS. Software included for compiling PAL source codes.

Software Available for CPM,[1] ISIS,[2] TRSDOS,[3] MSDOS.[4]

1. TM of Digital Research Corp.
2. TM of Intel Corp.
3. TM of Tandy Corp.
4. TM of Microsoft.

**Post Office Box 289**
**Waveland, Mississippi 39576**
**[601]-467-8048**

**$879 stand alone**
**MODEL 7956**

**MODEL 7956**
**GANG PROGRAMMER**
Intelligent algorithm. Stand alone, copies eight EPROMS at a time. With RS-232 option $1099.

**$549**
**MODEL**
**7316**

**$1195**
**MODEL**
**7324**

**MODEL 7324 PAL PROGRAMMER**
Programs all series 20 & 24 PALS. Operates stand alone or via RS232.

Avocet Cross Assemblers are available to handle 8748, 8751, Z8, 6502, 680X, etc. Available for CP/M and MSDOS computers. Order by processor type and specify kind of computer.

Model DE-4 U/V Products hold 8, 28 pin parts. High quality professional construction.

| | |
|---|---|
| Model 7128-L1,L2,L2A | $239.00 |
| Model 7128-24 | $329.00 |
| DR8 or DR5 | $ 30.00 |
| DR8PGX or DR5PGX | $ 75.00 |
| Cross Assemblers | $200.00 |
| XASM (for MSDOS) | $250.00 |
| U/V Eraser DE-4 | $ 78.00 |
| RS232 Cables | $ 30.00 |
| 8751 adapter | $174.00 |
| 8755 adapter | $135.00 |
| 48 Family adapter | $ 98.00 |

**$549**
**MODEL**
**7228**

**MODEL 7228**
**EPROM PROGRAMMER**
All features of Model 7128 plus Auto Select Baud , super fast adaptive programming algorithms, low profile aluminum enclosure. Programs 2764 in one minute!

**$429**
**MODEL**
**7128**

**MODEL 7128 EPROM PROGRAMMER**
**Programs and Read:**

| NMOS | NMOS | CMOS | EEPROM | MPU'S |
|---|---|---|---|---|
| 2508 | 2758 | 27C16 | 5213 | 8748 |
| 2516 | 2716 | 27C32 | 5213H | 8748H |
| 2532 | 2732 | C6716 | X2816 | 8749H |
| 2564 | 2732A | 27C54 | 48016 | 8741 |
| 68766 | 2764 | | 12816A | 8742H |
| 68764 | 27128 | | | 8741H |
| 8755 | 27256 | | | 8751 |
| 5133 | | | | |

Circle **no. 30** on reader service card.

P (Listing Continued, text begins on page 46)

## Listing One

```
        {       fputs(l_str(filenms), stderr);
                if (lineno)
                        fputs(", ", stderr);
                else
                        fputs(": ", stderr);
        }
        if (lineno)
        {       fputs("line ", stderr);
                fputs(itod(lineno), stderr);
                fputs(": ", stderr);
        }
        while (narg)
        {       fputs(*--argv, stderr);
                if (--narg)
                        fputc(' ', stderr);
        }
        fputc('\n', stderr);
}

kind(plp)               /* determine command */
                        /* move line pointer past it and white space */
        int *plp;       /* char**, -> line pointer; NULLed or advanced */
{       char *s;

        for (s = line+1; isspace(*s); ++s)
                ;
        if (*plp = cmd(s, "define"))    return DEFINE;
        if (*plp = cmd(s, "else"))      return ELSE;
        if (*plp = cmd(s, "endif"))     return ENDIF;
        if (*plp = cmd(s, "ifdef"))     return IFDEF;
        if (*plp = cmd(s, "ifndef"))    return IFNDEF;
        if (*plp = cmd(s, "include"))   return INCLUDE;
        if (*plp = cmd(s, "line"))      return LINE;
        if (*plp = cmd(s, "undef"))     return UNDEF;
        return DEFAULT; /* *plp is NULL */
}

cmd(l, c)               /* parse keyword */
                        /* return NULL or -> past it and white space */
        char *l;        /* -> begin of possible keyword */
        char *c;        /* -> keyword */
{
        /* compare */
        while (*l++ == *c++ && *c)
                ;
        if (*c)
                return 0;       /* incomplete keyword */
        if (*l == '\0')
                return 1;       /* just keyword */
        if (! isspace(*l))
                return 0;       /* keyword plus trash */
        while (isspace(*++l))
                ;
        return l;               /* bypassed white space */
}

markfl(sp)              /* bypass and terminate file name */
                        /* return true if found */
        char *sp;       /* -> begin delimeter, " or < */
```

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS
**Dr. Dobb's Journal** — **Reader Service Card**

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. This card is valid for 90 days from issue date. Use only one card per person.

July 1984, No. 93

Name _____ Phone (___) _____

Address _____

City/State/Zip _____

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 |
| 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 |
| 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 | 128 | 129 | 130 | 131 | 132 | 133 | 134 | 135 |
| Articles: | 190 | 191 | 192 | 193 | 194 | 195 | 196 | 197 | 198 | 199 | 200 | 201 | 202 | 203 | 204 | 205 | 206 | 207 | 208 | 209 | 210 | 211 | 212 | 213 | 214 | |

**Comments:**

**Purchase of Magazine:**

1 ☐ Subscription
2 ☐ Computer Store
3 ☐ Newsstand
4 ☐ Bookstore
5 ☐ Passed on by friend/colleague
6 ☐ Other _____


SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS
Dr. Dobb's Journal — Reader Service Card

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. This card is valid for 90 days from issue date. Use only one card per person.

July 1984, No. 93

Name _____ Phone (___) _____

Address _____

City/State/Zip _____

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 |
| 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 |
| 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 | 128 | 129 | 130 | 131 | 132 | 133 | 134 | 135 |
| Articles: | 190 | 191 | 192 | 193 | 194 | 195 | 196 | 197 | 198 | 199 | 200 | 201 | 202 | 203 | 204 | 205 | 206 | 207 | 208 | 209 | 210 | 211 | 212 | 213 | 214 | |

**Comments:**

**Purchase of Magazine:**

1 ☐ Subscription
2 ☐ Computer Store
3 ☐ Newsstand
4 ☐ Bookstore
5 ☐ Passed on by friend/colleague
6 ☐ Other _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

**BUSINESS REPLY CARD**
FIRST CLASS PERMIT NO. 756 MENLO PARK, CA

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS
**Dr. Dobb's Journal**

P. O. Box 27809
San Diego, CA 92128

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 756 MENLO PARK, CA

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS
Dr. Dobb's Journal

2464 EMBARCADERO WAY
PALO ALTO, CA 94303

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 756 MENLO PARK, CA

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS
Dr. Dobb's Journal

2464 EMBARCADERO WAY
PALO ALTO, CA 94303

```
{       char s, *cp, c;


        if ((s = *(cp = sp)) && (s == '\"' || s == '('))
                while (c = *++cp)
                        if (c == '\"' && s == '\"'
                            || c == ')' && s == '(')
                        {       *cp = '\0';
                                return cp-sp ) 1;
                        }
        return 0;
}
```

End Listing One

## Listing Two

```
/****
 ****    UN*X compatible dynamic memory allocation
 ****/

/*
 *      calloc  return pointer to vector of 0, or NULL
 *      cfree   free previously allocated area
 *
 *      The heap starts at _end and runs upward toward the stack.
 *      Each area in the heap is preceded by a word at an even address;
 *      a pointer chain runs from _end through these words to NULL;
 *      The low bit in each word is 1 if the following area is free.
 *      There is a blind, allocated element at the front of the chain.
 *
 *      BUG:    very unreasonable demands (e.g., wraparound)
 *              will corrupt memory.
 */

#define SLACK   1024    /* at least 1KB stack to be free */
#define NULL    0

word(wp)
        int *wp;
{
        return *wp;
}

char * calloc(n,len)
        int n;          /* number of elements */
        int len;        /* length of element */
{       int cell;       /* current allocation chain cell */
        char *p;        /* -) cell */
        char *np;       /* pointer in cell */
        int *ip, *wp;   /* for casting */

        len = (len*n + 1) & ~1;         /* even */
        if (len == 0)
                return NULL;
        for (ip = p = word(_end+1 & ~1) & ~1;
             np = (cell = *ip) & ~1;
             ip = p = np)
                if (cell & 1)           /* lowbit == 1 means free */
                {       if ((n = np-p - 2) ) len+2)
                        {       wp = p + len+2;
                                *wp = cell;
```

*(Continued on next page)*

**P** (Listing Continued, text begins on page 46)

## Listing Two

```
                        *ip = wp;
                }
                else if (n >= len)
                        *ip = np;
                else
                        continue;
                for (wp = p+2; len; len -= 2)
                        *wp++ = 0;
                return p+2;
        }
        if ((wp = p + len+2) > &n - SLACK)
                return NULL;
        *ip = wp;
        *wp = NULL;
        for (wp = p+2; len; len -= 2)
                *wp++ = 0;
        return p+2;
}

cfree(fp)
        int *fp;        /* to be freed */
{       int *p, *np;

        --fp;                           /* to cell */
        for (p = _end+1 & ~1;
             np = word(p) & ~1;
             p = np)                    /* p-> previous cell */
                if (np == fp)           /* fp-> cell to free */
                {       np = *fp;       /* np-> following cell */
                        if ((*fp & 1) || np == NULL)
                                break;  /* he does not own it */
                        if (*p & 1)
                                if (*np & 1)
                                        *p = *np;
                                else if (*np == NULL)
                                        *p = NULL;
                                else
                                {       *p = np;
                                        *p |= 1;
                                }
                        else if (*np & 1)
                                *fp = *np;
                        else if (*np == NULL)
                                *fp = NULL;
                        else
                                *fp |= 1;
                        return;
                }
        fputs("cfree botch", stderr);
        exit();
}
```

**End Listings**

# MagiKey★

**THE FULL-FEATURED KEYBOARD EXPANDER**
**for all 8080-8085-Z80 computers using CP/M 2.2**

**MagiKey™** will redefine your keys as character strings ... and transform single keystrokes into commands for programs, words for word processors, data for data bases, and messages for modems. Without hardware or system modifications.

**MagiKey™ has more advanced features than any other CP/M keyboard expander.** For example:

★ Redefine a key to send the string:
"Run WordStar and edit form letter number 17"
Hit the key. YOU will see the string, but "WS FRMLTR17" will be sent to CP/M. **MagiKey™**'s CONSOLE REDIRECTION can display messages which are invisible to programs and CP/M ... very useful for recalling key assignments, custom operator prompts, and making CP/M friendlier.

★ Redefine a key to run several programs in sequence. Each program can wait for keyboard input or receive pre-defined commands and data. **MagiKey™**'s built-in BATCH PROCESSING doesn't use CP/M's SUBMIT, and handles programs that CP/M's XSUB can't.

★ Redefine a key to display the prompt:
"Execute SuperCalc using the spreadsheet file:"
Press the key to display the prompt. Type the file name, hit RETURN, and you're into SuperCalc. When done, a single keystroke saves your updated spreadsheet. You don't have to remember or retype its name. **MagiKey™**'s RECURSIVE key redefinition mode automatically does it for you.

**WE INVITE COMPARISON**

**$100**
8" SSSD, most 5¼" formats
add 6% tax in CA
check, VISA, M/C

CP/M (tm) Digital Research
SuperCalc (tm) Sorcim
WordStar (tm) Micropro

PRO microSystems
16609 Sagewood Lane
Poway, California 92064
(619) 693-1022

Circle **no. 54** on reader service card.

# Q/C

For only $95, Q/C is a ready-to-use C compiler for CP/M. You get complete source code for the compiler and over 75 library functions. Q/C is upward compatible with UNIX Version 7 C, but doesn't support long integers, float, parameterized #defines, and bit fields.

- Full source code for compiler and library.
- No license fees for object code.
- Z80 version takes advantage of Z80 instructions.
- Excellent support for assembly language and ROMs.
- Q/C is standard. Good portability to UNIX.

Version 3.2 of Q/C has many new features: structure initialization, faster runtime routines, faster compilation, and improved ROM support. Yes, Q/C has casts, typedef, sizeof, and function typing. The *Q/C User's Manual* is available for $20 (applies toward purchase). VISA and MasterCard welcome.

THE CODE WORKS

5266 Hollister
Suite 224
Santa Barbara, CA 93111
(805) 683-1585

Q/C, CP/M, Z80, and UNIX are trademarks of Quality Computer Systems, Digital Research, Zilog, Inc., and Bell Laboratories respectively.

Circle **no. 14** on reader service card.

# Six Times Faster!

## Super Fast Z80 Assembly Language Development Package

### Z80ASM

- Complete Zilog Mnemonic set
- Full Macro facility
- Plain English error messages
- One or two pass operation
- Over 6000 lines/minute
- Supports nested INCLUDE files
- Allows external bytes, words, and expressions (EXT1 * EXT2)
- Labels significant to 16 characters even on externals (SLR Format Only)
- Integral cross-reference
- Upper/lower case optionally significant
- Conditional assembly
- Assemble code for execution at another address (PHASE & DEPHASE)
- Generates COM, HEX, or REL files
- COM files may start at other than 100H
- REL files may be in Microsoft format or SLR format
- Separate PROG, DATA & COMMON address spaces
- Accepts symbol definitions from the console
- Flexible listing facility includes TIME and DATE in listing (CP/M Plus Only)

### SLRNK

- Links any combination of SLR format and Microsoft format REL files
- One or two pass operation allows output files up to 64K
- Generates HEX or COM files
- User may specify PROG, DATA, and COMMON loading addresses
- COM may start at other than 100H
- HEX files do not fill empty address space.
- Generate inter-module cross-reference and load map
- Save symbol table to disk in REL format for use in overlay generation
- Declare entry points from console
- The FASTEST Microsoft Compatible Linker available


- Complete Package Includes: Z80ASM, SLRNK, SLRIB - Librarian and Manual for just $199.99. Manual only, $30.
- Most formats available for Z80 CP/M, CDOS, & TURBODOS
- Terms: add $3 shipping US, others $7. PA add 6% sales tax

For more information or to order, call:

**1-800-833-3061**

In PA, (412) 282-0864

Or write: SLR SYSTEMS
1622 North Main Street, Butler, Pennsylvania 16001

SLR Systems

Circle **no. 66** on reader service card.

# A Simple Minimax Algorithm

by Steven A. Ruzinsky

*Steven Ruzinsky, 2110 S. Austin Blvd., Cicero, IL 60650*

Computers frequently use polynomial or rational approximations for transcendental functions, and yet the typical programmer's involvement with these approximations is limited simply to looking up the appropriate parameters of the approximation in a handbook. Have you ever wondered where these approximations came from? Is it easy for programmers with some experience in numerical analysis to invent their own approximations? Suppose you need an approximation that is unavailable in the literature — what do you do?

Possibly these questions have never crossed your mind because you just assumed that these approximations represent a simple Least Squares (LS) fit. I'll tell you right now, these approximations are not a LS fit. Usually they represent a Minimum Maximum Absolute Error (Minimax) fit; that is, rather than determining the approximation parameters so that the mean-squared error or error variance is a minimum over the interval of approximation, the parameters were determined so that the largest absolute value of the errors encountered over the interval of approximation is a minimum. In other words, the worst error has been minimized.

Synonyms for Minimax approximation include "Chebyshev approximation" and "$\ell_\infty$ approximation." Sometimes, particularly in statistical works, you will see the word "estimate" in place of "approximation." Both LS and Minimax belong to a larger class of approximations that minimize a norm of errors, the particular type of norm defining the type of approximation. Of all the approximations in this class, LS is by far the most popular, most easily calculated, and most discussed.

Because many LS principles carry over to the case of Minimax, this article assumes that the reader has some hands-on experience with LS curve fitting. If you have no such experience and are still interested in Minimax, I suggest that you first acquire some, starting with the vast body of literature on LS.

The reason for choosing Minimax criterion over others such as LS should be obvious. We want to guarantee that the errors associated with an approximation fall within specified limits and to make these limits as small as possible without increasing the amount of time and memory used in calculation. An approximation has two sources of calculation parsimony. First is the form of the approximation. Different forms include odd polynomials, even polynomials, rational functions (quotients of polynomials), and so on. That the selection of form is an art and not a science makes such selection very interesting, but unfortunately that is outside the scope of this article. The second source of parsimony is the type of fit that the approximation uses. For a given form of approximation, the Minimax fit is always the most parsimonious.

In spite of the fact that Minimax approximations are fundamental to computerdom, the techniques for Minimax curve fitting are esoteric, probably because traditional methods are somewhat complicated, often not automatic, and therefore difficult to implement as computer programs. Using conventional techniques, programmers would find it difficult to invent their own approximations. I refer those of you wishing to find out more about these techniques to the work of Hastings,[1] Scheid,[2] and Dem'yanov and Malozemov.[3] The latter provides a rigorous mathematical treatment of the subject and outlines a large number of Minimax algorithms, which, unfortunately, are far removed from the cookbook style that programmers are likely to appreciate. Scheid walks his readers through two of the most popular algorithms, the Exchange Method and a method involving Simplex linear programming, with numerous pencil and paper examples — I highly recommend it. Hastings is the best text for learning the artistic aspects of curve fitting, such as form selection.

To the best of my knowledge, however, this article represents the first publication of actual programs for Minimax curve fitting. I propose to introduce an original Minimax algorithm that offers the advantages of simplicity and accuracy over conventional methods.

With respect to simplicity, this algorithm should result in a minimum of program source code. Some conventional methods, such as the Exchange Method, look deceptively simple until you actually try programming them; the Exchange Method involves data transfers between matrices that are quite messy to program. I challenge anyone to program a conventional Minimax algorithm (with the same degree of functionality) in less than *twice* the source code of the BASIC program of Listing One (page 98).

In using any Minimax algorithm, as you seek higher and higher order approximations, you eventually reach an order where the algorithm fails due to an accumulation of roundoff errors. My algorithm should identify approximations of a higher order (i.e., with more parameters) than any other algorithm using the same precision arithmetic. I have three reasons for believing this:

(1) This algorithm involves the solution of linear equations of order n, whereas other algorithms involve solutions of higher order equations; in the

case of the Exchange Method, this order is n + 1 (n is the order of the approximation).
(2) This algorithm solves these linear equations via the Sequential Least Squares algorithm (described later), which is relatively immune to ill-conditioned systems of equations.
(3) To a limited extent, each iteration within the algorithm is compensatory toward roundoff errors incurred from previous iterations, while other algorithms, although iterative, do not have this compensatory effect.

I would be surprised if you can find a Minimax algorithm that will identify higher order approximations than those of the Forth Matrix Language (FML) program of Listing Two (page 99) using double-precision (64-bit) floating point arithmetic. Figures 4–21 (pages 93–97) show the results of the FML program.

### Conventions

Before getting into matrix equations, I should define some conventions. Lower-case letters without underscores represent scalar variables. Lower-case letters with underscores represent column vectors. Upper-case letters represent matrices other than vectors. The superscript T indicates matrix transpose. The caret (^) over a variable indicates that it is an approximation or estimate of the variable with the corresponding symbol. The "order" of an approximation is the total number of parameters to be identified, whereas the "degree" of a polynomial is the highest power contained in that polynomial: thus, $y = a_1 x^{10}$ is a polynomial of degree 10 and order 1.

### The Error-Fluffing Algorithm

The use of this algorithm is not limited to curve fitting but is applicable to any system that is linear in its parameters. Consider the most general case:

$$y = a_1x_1 + a_2x_2 \cdots + a_nx_n + e \qquad (1)$$

or in matrix notation $y = \underline{a}^T\underline{x} + e$

where

$$\underline{a} \equiv \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ . \\ a_n \end{bmatrix}, \quad \underline{x} \equiv \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ . \\ x_n \end{bmatrix}, \text{ and } e \equiv \text{Error}$$

which we wish to approximate with

$$\hat{y} = a_1x_1 + a_2x_2 \cdots + a_nx_n \qquad (2)$$

or

$$\hat{y} = \underline{a}^T\underline{x}$$

I anticipate that some people familiar only with the statistical application of LS regression may become confused at this point. Although the error, e, in Equation (1) can be stochastic, in all of the examples used here the error, e, will be completely deterministic. The Minimax algorithm uses LS algorithms, which are derived in the Appendix, without any assumption of the stochasticity or determinability of e.

Given a set of m data points:

$$\{(\underline{x}_i, y_i)\}_m, \; m \geq n,$$

this algorithm iteratively determines a second set of k data points:

$$\{(\underline{x}_i, y_i)\}_k, \; k \geq m,$$

and coefficients $\underline{a}$, such that $\underline{a}$ represents an LS fit to the second set of points and a Minimax fit to the first set of points. In the algorithm's simplest form, $\{(\underline{x}_i, y_i)\}_k$ is a superset of $\{(\underline{x}_i, y_i)\}_m$ containing only multiple replications of the points in $\{(\underline{x}_i, y_i)\}_m$.
A good way to conceptualize this is to consider the first set of points as being locations $(\underline{x},y)$ space and the second set as having points at those locations, with many of the locations having multiple points from the second set.

The procedure, simplified to facilitate understanding, is:
Let k = iteration index, and
(1) Use an LS fit as the initial estimate of $\underline{a}_0$.
(2) Find the maximum absolute error resulting from $\underline{a}_k$.
(3) Reincorporate the point at which the maximum absolute error occurs back into the data set.
(4) Perform LS regression on the new data set.
(5) Repeat, starting at Step 2, until no further decrease in maximum absolute error seems likely with further iteration (convergence is nonmonotone).

Since a rigorous mathematical convergence proof of this algorithm is presently unknown, I cannot guarantee that it will always work. However, except for failures due to roundoff errors, I have yet to encounter a single case where this algorithm didn't work! In a practical sense, this Minimax algorithm is more foolproof than LS algorithms because we can easily check to see if the results are truly Minimax. The error curve (i.e., e versus x) of a Minimax fit will contain at least n + 1 minima and maxima that are equal in absolute value; that is, at least n + 1 equal worst errors will appear. Remember, n is the number of estimated parameters and not necessarily the degree of a polynomial. In almost all cases of practical interest, the error curve will have exactly n + 1 equal and unique minima and maxima; under this condition, we can be reasonably certain that the fit is Minimax. By contrast, LS algorithms, which have a firm mathematical foundation, frequently generate erroneous results (because of roundoff errors) that go undetected — there is no easy way to check a LS result.

Although I lack a convergence proof of the Minimax algorithm, you should be able to understand at an intuitive level how the algorithm works. Replicating the data point at which the maximum absolute error occurs gives increased weight to the location of that point; that is, to minimize the error variance with respect to the new set of points, the error at that location will be reduced while increasing elsewhere. As this process repeats, the maximum absolute error eventually appears at a new location. This new location receives the same treatment, but now previously weighted locations will resist error increases. Because this constrains the maximum absolute error, it will nonmonotonically approach a minimum. Incidentally, it is this nonmonotone nature of the algorithm that makes a mathematical convergence proof difficult.

It is very interesting to watch the change in the error curve with each iteration. Because the process resembles fluffing the bulges out of a pillow, I use the term "error fluffing" to describe this algorithm.

### The SLS Algorithm

You should have noticed two impractical aspects to the five-step simplified procedure. First, with every iteration,

Step 3 adds one point to the data set. Since thousands of iterations may be required, this growing set of data points could easily exceed the memory capacity of even the largest mainframe! Second, in Step 4, performing LS regression on even a moderately large set of points is very time-consuming.

Fortunately, we can combine Steps 3 and 4 so that the set $\{(\underline{x}_i, y_i)\}_k$ exists only as a mathematical abstraction without occupying any RAM; remember, we are only interested in getting $\underline{a}_k$, the coefficients corresponding to an LS fit to $\{(\underline{x}_i, y_i)\}_k$

The method for doing this, which is also very fast, is called the Sequential Least Squares (SLS) algorithm. Indeed, without the SLS algorithm, this Minimax algorithm would be of no practical value. Using the SLS algorithm, the time per iteration for this Minimax algorithm is significantly less than that of other iterative algorithms such as the Exchange Method. Thus, even though such algorithms may converge in less iterations, my algorithm may be as fast!

I would like to digress a moment and say that the SLS algorithm is the most wasted mathematical resource I know. Outside of the electrical engineering community, this algorithm is almost unknown. For example, of the numerous statistical software packages on the market, to the best of my knowledge none uses the SLS algorithm. The SLS algorithm, which resembles a Kalman filter, was once used for real-time analysis of time series (e.g., in speech recognition), but recently the faster "lattice" or "ladder" algorithms have replaced it. Use of these newer algorithms, however, is limited to identifying ARMA (AutoRegressive-Moving Average) parameters for time series, whereas the SLS algorithm is applicable to *all* forms of linear regression.

The SLS algorithm is also particularly well suited for a large variety of microcomputer applications because it conserves memory and is, without any doubt, the easiest LS algorithm to program. The SLS algorithm avoids matrix inversion and involves little more than matrix multiplication and addition. By letting m = n, you can even use the SLS algorithm in place of such algorithms as the Gaussian Elimination to solve n linear equations in n unknowns or to invert a matrix! I have included a complete description and derivation of the SLS algorithm in the Appendix. For further information on LS algorithms and their applications to time series analysis, see the two books by Graupe.[4,5]

## Implementation

The two implementations of the Minimax algorithm include one in BASIC (Listing One) and another in Polyforth with FML (Listing Two). FML is an APL-like extension to Forth developed by this author, which, when used with the 8087, I believe to be the fastest numerical language available for microcomputers. Version 1.0 of FML is in the public domain and can be found in Nos. 80, 81, and 82 of *Dr. Dobb's Journal*. The illustrative program presented here is written in the commercially available version 3.0. Besides speed, the main advantage of FML is that it performs matrix operations with syntactically simple statements, thus facilitating quick and convenient "on the fly" programming of complex mathematical algorithms; for example, LS regression is performed by a single word in version 3.0. FML also allows arrays of up to 256Kbytes, thus allowing storage of huge data sets.

Such being the case, some drastic differences exist between the FML and BASIC implementations. Because the BASIC program represents the algorithm in its simplest possible form, it will not be able to duplicate the performance of the FML program in regard to either speed or accuracy. The BASIC program is described in the comment fields, and major differences between the BASIC and FML programs are mentioned in the text describing the FML program. If anyone is interested in the nitty-gritty details of how the BASIC program works, I suggest that you first study the FML program description, then the SLS section of the Appendix, and last the comment field of the BASIC program.

The BASIC program employs an example used by both Hastings[1] (page 138) and Ruckdeschel[6] (page 522). Here Equation (1) takes the form:

$$y = a_1x + a_2x^3 + a_3x^5 + e$$

where

$$y = \text{SIN}(PI*X/2)$$

However, in this particular instance, $\underline{a}$ is determined to a Minimum Maximum Absolute *Relative* Error criterion. Dividing both sides of the above equation by y:

$$(1) = a_1(x/y) + a_2(x^3/y) + a_3(x^5/y) + (e/y)$$

and redefining y, $\underline{x}$, and e to be the bracketed quantities puts Equation (1) in a form so that $\underline{a}$ will be determined in a way to minimize the relative error, (e/y). The coefficients resulting from this program are $a_1 = 1.5706258$, $a_2 = -0.6432224$, and $a_3 = 0.0727045$; the resulting maximum absolute relative error is 0.00010856. Hastings' result is $a_1 = 1.5706268$, $a_2 = -0.6432292$, and $a_3 = 0.0727102$, with a maximum absolute relative error of 0.00010879.

Figure 1 (page 92) shows the error curve for a Minimax fit, whereas Figure 2 (page 92) shows the curve for an LS fit. The similarities and differences between these two error curves are typical of LS and Minimax fits in general. Notice that the worst error of the LS fit occurs near the range ends and that the worst error of the Minimax fit is 55% smaller. Ruckdeschel attempted a Minimax fit using a very general optimization by steepest descent algorithm; Figure 3 (page 92) shows the resulting error curve. Clearly, his fit is not Minimax, thereby demonstrating the inadequacy of gradient algorithms for this purpose.

You can easily modify the BASIC program to find other approximations. For example, setting N = 4 in line 140 will result in a fourth order approximation.

## The FML Implementation

Screen 207 contains the final FML program, formatted as a list of steps. I do not expect you to be able to read FML. A detailed description of the steps follows, so that no one should have much difficulty in implementing the algorithm in the language of their choice.

### *Insert Data*

First, the set of sample points

$$\{(\underline{x}_i, y_i)\}_k$$

is entered into matrices and X as follows:

$$\underline{y} \equiv \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ . \\ y_m \end{bmatrix}$$

and

$$X \equiv \begin{bmatrix} \underline{x}_1^T \\ \underline{x}_2^T \\ \underline{x}_3^T \\ . \\ \underline{x}_m^T \end{bmatrix} \equiv \begin{bmatrix} x_{11}\ x_{12}\ x_{13} \cdots x_{1n} \\ x_{21}\ x_{22}\ x_{23} \cdots x_{2n} \\ x_{31}\ x_{32}\ x_{33} \cdots x_{3n} \\ . \quad . \quad . \quad \cdots . \\ x_{m1} x_{m2} x_{m3} \cdots x_{mn} \end{bmatrix}$$

You should realize that the Minimax algorithm is applicable only to a finite number of points and not to the actual function, which is continuous. Because of this, we must hope that the resulting approximation will accurately interpolate between the sample points of the function being approximated. Generally, inspection of the error curve and the use of common sense are all that is needed to verify that the number of points, m, is sufficient.

As a rule, approximations with larger numbers of parameters will also require more points because the peaks and valleys in the error curve will be sharper and therefore more likely to protrude between sample points. If these peaks and valleys are crowded (hence, more acute at one or both ends of the interval of approximation), you then can select samples of x that are crowded at the ends. Note that in this respect Minimax is totally different from LS. To achieve an LS approximation to a continuous function, you generally must select points at equal intervals of x.

### Batch Least Squares

Next, the LS coefficients, $\underline{a}_o$, are calculated:

$$Q = (X^T X)^{-1} \qquad (3)$$

$$\underline{a}_0 = Q(X^T \underline{y}) \qquad (4)$$

Q, an n x n matrix, is called the "inverse covariance matrix." The parentheses in Equation (4) indicate the preferred order of calculation. Actually, you can substitute almost any LS algorithm, including those that do not use matrix inversion, for Equation (4), but it is necessary to calculate Q, Equation (3), because Q will be used later. (The BASIC program uses the SLS algorithm in place of Equations (3) and (4). Using SLS from a cold start introduces an error into the calculations, as explained in the Appendix.)

### Reduce Roundoff Error

Large roundoff errors often result from LS algorithms such as those of Equations (3) and (4). Ruckdeschel describes a method to reduce these errors. First, the error vector, e, is calculated:

$$\underline{e} \equiv \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ . \\ e_m \end{bmatrix} = \underline{y} - X\underline{a} \qquad (5)$$

If we redesignate the old coefficient vector as, a′, the improved estimate, is:

$$\underline{a} = \underline{a}' + Q(X^T \underline{e}) \qquad (6)$$

Equations (5) and (6) should be repeated iteratively, and the best $\underline{a}$ (i.e., the $\underline{a}$ corresponding to the smallest variance) saved. Equation (6) is tantamount to LS regression on the error, $\underline{e}$, adding the resulting coefficients to $\underline{a}$. Again, you can substitute almost any other LS algorithm in this step. (The BASIC program skips this step.)

### Normalize Q

The Q matrix is used in succeeding calculations. Although the Minimax algorithm will work with Q as is, increasing the values in Q by a constant factor will hasten convergence considerably. Beyond a certain limit, however, convergence may be lost completely. I found a good scaling factor to be m/10n. If we redesignate the old matrix as Q′, the new matrix will be:

$$Q = (m/10n)Q'$$

(The BASIC program skips this step.)

The following operations are reiterated, k being the iteration index; ultimately, $\underline{a}$, will converge to a Minimax fit.

### Find MAX(e), xk; Record Best a, ek

First, Equation (5) is used to calculate $\underline{e}$, then the maximum absolute error,

$|e_k|$ is found, and $e_k$ and the corresponding $\underline{x}_k$ from the matrix X are recorded for future use. Since convergence is nonmonotone, the best $|e_k|$ among all iterations to this point in time and the corresponding $\underline{a}_k$ should be recorded. After iteration is terminated, this best $\underline{a}_k$ is used as the final result.

### *Sequential Least Squares*

As previously mentioned, the SLS algorithm is the key to making this Minimax algorithm practical. The SLS algorithm adds a data point by updating the Q matrix and $\underline{a}_k$ without updating X and $\underline{y}$. Thus, X and $\underline{y}$ do not grow in dimension, and we can avoid the time-consuming operations represented by Equations (3), (4), and (6):

$$Q_k = Q_{k-1} - \frac{Q_{k-1}\,\underline{x}_k\,\underline{x}_k^T\,Q_{k-1}}{1 + \underline{x}_k^T\,Q_{k-1}\,\underline{x}_k} \qquad (8a)$$

$$\underline{a}_k = \underline{a}_{k-1} + Q_k\,\underline{x}_k\,(y_k - \underline{x}_k^T\,\underline{a}_{k-1}) \qquad (8b)$$

or: $\underline{a}_k = \underline{a}_{k-1} + Q_k\,\underline{x}_k\,e_k$

Notice that only in the context of Minimax do ( $\underline{x}_k, y_k$ ), and $e_k$ represent the data point and error corresponding to $|e_k|$ the maximum absolute error that results from $\underline{a}_{k-1}$ .

### *Speedup*

The SLS algorithm is fast and contributes negligible time to the Minimax algorithm. The bottleneck here is in the calculation of $\underline{e}$ with each iteration. After a certain number of iterations, however, we may use the fact that the maximum absolute errors now appear at only a small subset of the original sample points to hasten the process. The method used here consists of sorting the rows of $\underline{y}$ and X with respect to a descending order of absolute values of the elements of $\underline{e}$ then halving the dimensions, m, of, $\underline{y}$, $\underline{e}$, and X by truncation. (The BASIC program skips this step.)

### The Results

Now that we've finished describing the Minimax algorithm in FML, let's look at some results. Figures 4–21 show the parameters and resulting error curves of approximations to nine different functions. All of these approximations were found with the FML program. The even-numbered figures duplicate (with greater accuracy) the highest order results of Hastings for the indicated functions. The odd-numbered figures represent the highest order fits that I was able to achieve for those same functions. In many instances, the program identified more than twice Hastings' number of parameters. The table (at left) compares the accuracy of various approximations.

You should notice that the approximations given in Figures 10, 11, 12, 13, 18, 19, 20, and 21 are not linear in their parameters. How, then, did I manage to use my algorithm on them? There are a number of tricks for adapting linear algorithms to nonlinear situations. In the above cases, a simple transformation changes these approximations into the form of Equation (1) so that the algorithm can be successfully applied. As a contest, a free one-year subscription to *Dr. Dobb's* will be awarded to the first person to correctly describe this transformation.

Now to answer the question: "How fast is this Minimax algorithm?" A suitable reference for comparison is the time taken by a Batch LS algorithm to fit the same data. For a case with n = 12 and m = 320, the Batch LS portion of the FML program takes 2.37 sec, whereas each iteration of the loop of the Minimax algorithm takes 0.53 sec. (before SPEEDUP). As previously mentioned, the time contribution of the SLS algorithm, 0.08 sec per iteration, is negligible. The calculation of

| FUNCTION | FIGURE | NUMBER OF PARAMETERS | MAXIMUM ABSOLUTE ERROR | PERCENT IMPROVEMENT OVER HASTINGS' RESULT | PERCENT IMPROVEMENT OVER LEAST SQUARES |
|---|---|---|---|---|---|
| $LOG_{10}(X)$ | 4 | 5 | 1.32254 E-07 | 0.8 | 56 |
| | 5 | 12 | 1.11137 E-15 | | 62 |
| ARCTAN(X) | 6 | 8 | 3.75120 E-08 | 0.7 | 63 |
| | 7 | 12 | 2.25114 E-11 | | 42 |
| $SIN\left(\frac{\pi}{2}x\right)$ | 1 | 3 | 1.08179 E-04* | 0.6 | 55 |
| | 8 | 5 | 5.31748 E-09* | 0.9 | 62 |
| | 9 | 7 | 6.28159 E-14* | | 66 |
| $10^x$ | 10 | 7 | 4.96196 E-09* | 0.9 | |
| | 11 | 10 | 1.18194 E-13* | | |
| $\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$ | 12 | 6 | 2.21756 E-04 | 0.1 | |
| | 13 | 16 | 2.32245 E-09 | | |
| ARCSIN(X) | 14 | 8 | 2.18036 E-08 | 0.6 | 62 |
| | 15 | 13 | 1.60434 E-12 | | 67 |
| LN(1+X) | 16 | 8 | 3.21271 E-08 | 0.3 | 52 |
| | 17 | 13 | 2.99006 E-12 | | 61 |
| $e^{-x}$ | 18 | 6 | 2.42142 E-07 | 0.1 | |
| | 19 | 12 | 3.42650 E-13 | | |
| ERF(X) | 20 | 6 | 2.59522 E-07 | 0.3 | |
| | 21 | 14 | 1.41491 E-12 | | |

*RELATIVE ERROR

**Table**

**Comparison of Minimax Results with that of Hastings and LS**

# BACK ISSUES

**#66 Volume VII, No. 4:**
8080-Z80/8086 Cross-Assembler, Part 2—Writing the Runic Compiler—Poor Person's Spelling Checker.

**#68 Volume VII, No. 6:**
Multi-68000 Personal Computer— PDP-1802, Part One—Improved LET for LLL Basic—CP/M Print Utility.

**#69 Volume VII, NO. 7:**
IBM-PC Issue! CP/M-86 vs. MSDOS (A Technical Comparison)—Hi-Res Graphics on the IBM-PC—PDP-1802, Part II—Review of Word Processors for IBM.

**#70 Volume VII, NO. 8:**
Argum "C" Command Line Processor—SEND/RECEIVE File Transfer Utilities—Intel's 8087—Performance Evaluation.

**#71 Volume VII, No. 9:**
FORTH Issue! Floating-Point Package—H-19 Screen Editor—Relocating, Linking Loader—Z8000 Fourth—Fourth Programming Style—8086 ASCII-Binary Conversion Routines—CP/M Conditional SUBMIT.

**#72 Volume VII, NO. 10:**
Portable Pidgin for Z80—68000 Cross Assembler, Part I—MODEM and RCP/Ms—Simplified 68000 Mnemonics—Nested Submits—8086/88 Trig Lookup.

**#73 Volume VII, No. 11:**
Wildcard UNIX Filenames—Tests for Pidgin—68000 Cross Assembler Listing, Part 2—Adding More BDOS Calls—The Perfect Hash—BASIC Memory Management—Benchmarks for CP/M-86 vs. MSDOS, and the 8087.

**#76 Volume VIII, Issue 2:**
PISTOL, A Forth-like Portably Implemented STack Oriented Language—Program Linkage by Coroutines, Forth to BASIC—Linking CP/M Functions to Your High-Level Program—Concurrent CP/M-86—Small Systems Engineering CP/M-80 Expansion Card for the Victor 9000—REVAS Disassembler.

**#77 Volume VIII, Issue 3:**
Augusta, Part II: The Augusta P-Code Interpreter—A Small-C Operating System—6809 Threaded Code: Parametrization and Transfer of Control—A Common-Sense Guide to Faster, Small BASIC—A Fundamental Mistake in Compiler Design—Basic Disk I/O, Part I.

**#78 Volume VIII, Issue 4:**
RECLAIM Destroyed Directories—Binary Magic Numbers—8080 Fig-Forth Directory & File System—SAY" Forth Votrax Driver—TRS-80 8080 to Z80 Translator—Basic Disk I/O, Part II.

**#79 Volume VIII, No. 5:**
Augusta Part III: The Augusta Compiler—A Fast Circle Routine—Enhancing the C Screen Editor—Shifts and Rotations on the Z80—The SCB, TSX, and TXS Instructions of the 6502 and 6800—MS-DOS vs. CP/M-86—Controlling MBASIC—The Buffered Keyboard—IBM PC Character Set Linker—Flip Utility for the IBM PC.

**#80 Volume VIII, Issue 6:**
Fast Divisibility Algorithms—B-Tree ISAM Concepts—CP/M BDOS AND BIOS Calls for C—Serial Expansion in Forth—Fast Matrix Operations in Forth, Part I—Yes, You Can Trace Through BDOS—Julian Dates for Microcomputers—8088 Addressing Modes—8088 Line Generator—CP/M Plus.

**#81 Volume VIII, Issue 7:**
Augusta, Part IV (The Augusta Compiler, continued)—RED: A Better Screen Editor, Part I—Anatomy of a Digital Vector and Curve Generator—Fast Matrix Operations in Fourth, Part II—The AGGHHH! Program—MBOOT Revisited—CP/M Plus Feedback—MS-DOS Rebuttal—68000 Tools—Sizing Memory on the IBM PC.

**#82 Volume VIII, Issue 8:**
Serial-to-Parallel: A Flexible Utility Box—McWORDER: A Tiny Text Editor—And Still More Fifth Generation Computers!—Specialist Symbols and I/O Benchmarks for CP/M Plus—CP/M Plus Memory Management—Zero Length File Test—PAUSEIF, QUITIF, and now SKIPIF—ACTxx Cross Assemblers.

**#83 Volume VIII, No. 9:**
FORTH ISSUE! Forth and the Motorola 68000—Nondeterministic Control Words in Forth—A 68000 Forth Assembler—GO in Forth—Precompiled Forth Modules—Signed Integer Division—Some Forth Coding Standards—The Forth Sort—A speed and accuracy benchmark program for high-level languages—Using a Digital Spreadsheet Program for Something Fun and Unusual.

**#84 Volume VIII, No. 10:**
DDJ's new C/Unix column!—Unix to CP/M Floppy Disk File Conversion—A Small-C Help Facility—Attaching a Winchester Hard Disk to the S-100 Bus—Using Epson Bit-Plot Graphics—Your Fantasy Computer System—8086/88 Function Macros—Auto Disk Format Selection—CP/M Plus Device Tables.

**#85 Volume VIII, Issue 11:**
A Kernel for the MC68000—A DML Parser—Towards a More Writable Forth Syntax—Simple Graphics for Printer—8080 to Z80 Program Conversion, CP/M Plus DPB Macro Fix, and Quicker Submit File Truncation—Floating-Point Benchmarks and an MSDOS COM File Loader—software and book reviews.

**#86 Volume VIII, Issue 12:**
Faster Circles for Apples—Cursor Control for Dumb Terminals—Dysan's Digital Diagnostic Diskette—Building a Programmable Frequency Synthesizer—Unix and *Non*-Interactive, User-*Un*friendly Software—Interfacing a Hard Disk Within a CP/M Environment—The New MS-DOS EXEC Function.

**#87 Volume IX, Issue 1:**
NBASIC: A Structured Preprocessor for MBASIC—A Simple Window Package—Forth to PC-DOS Interface—Sorted Diskette Directory Listing for the IBM PC—Emulate WordStar on TOPS-20—More on optimising compilers—Problems under CP/M Plus with PAUSE—The PIP mystery device contest—Microsoft BASIC—An improved CLINK utility,

**#88 Volume IX, Issue 2:**
Telecommunications Issue! Micro to Mainframe Connection—Communications Protocols—Unix to Unix Network Utilities—VPC: A Virtual Personal Computer for Networks—PABX and the Personal Computer—BASIC Language Telecommunications Programming—U.S. Robotics S-100 Card Modem—Sysdrive program, bringing up CP/M Plus—PCDOS Close Function—The Microsoft Assembler—more on C layout standards—book and software reviews.

**#90 Vol. 9, Issue 4:**
Optimizing Strings in C—Expert Systems and the Weather—RSA: A Public Key Cryptography System, Part II—BASICFMT for TRS-80—Several items on CP/M Plus—CP/M v2.2 Compatibility—BDOS Function 10: Vastly Improved—More on MS-DOS EXEC Function—MS-DOS Volume Labels—Finding Size of TPA under MS-DOS 2.0—some comments about Unix—Low-Level Input-Output in C—More on Link Formats and Runtime Libraries—some information on TEX—software reviews.

**#91 Vol. 9, Issue 5:**
Introduction to Modula-2 for Pascal Programmers—Converting Fig-Forth to Forth-83—Sixth Generation Computers—A New Library for Small-C—The Accent Finder—Solutions to Quirks in dBASE II—Comments on optimizing compilers, Prolog, RMAC and MSDOS, IRD: contest results, and some CP/M Tidbits.

## TO ORDER:

Send $3.50 per issue to: **Dr. Dobb's Journal**, 2464 Embarcadero Way, Palo Alto, CA 94303

Please send me the issue(s) circled: **66 68 69 70 71 72 73 76 77 78 79 80 81 82 83 84 85 86 87 88 90 91**

I enclose $ _____ (U.S. check or money order).
Outside the U.S., add $.50 per issue.

Please charge my: ☐ Visa ☐ M/C ☐ /Amer. Exp.

Card No. _____

Exp. Date _____

Signature _____

Name _____

Address _____

City _____

State _____ Zip _____

Availability on first come/first serve basis. Outside the U.S., add $.50 per issue ordered. Price includes issue, handling, and shipment by second class or foreign surface mail. Within the U.S., please allow 6–9 weeks to process your order second class. For faster service within the U.S., we'll ship UPS if you *add* $1.00 for 1–2 issues and $.50 for *each issue* thereafter. We need a street address, not a P.O. Box. Outside the U.S., *add* $1.50 *per issue* requested for airmail shipment.

3BIJ

Circle **no. 83** on reader service card.

the $\underline{e}$ vector accounts for most of the 0.53 seconds.

Initial convergence is rapid, so that only 200–500 iterations may be required to achieve the accuracy obtained by Hastings. However, maximum accuracy is achieved at about 40,000 iterations; with double precision, roundoff errors prevent gains in accuracy with further iteration.

In summary, the Minimax algorithm presented here is simple enough for those interested in numerical analysis to use on a casual basis. With this algorithm and double-precision arithmetic, programmers can find Minimax approximations that are both of a higher order and more accurate than those published in the literature. Furthermore, the use of this algorithm is not limited to deterministic systems.

It may be interesting to explore the use of Minimax in statistics. Typically, LS is used because, for systems in which the error is random and normally distributed, LS either is or closely approximates a maximum likelihood estimator of the parameters. For certain systems where the error has a statistical distribution that is bounded, however, Minimax may be more statistically efficient.

## References

1. Hastings, Cecil, *Approximations for Digital Computers*, Princeton University Press, 1955.

2. Scheid, F., *Numerical Analysis, Schaum's Outline Series*, McGraw-Hill, 1968.

3. Dem'yanov, V. F., and Malozemov, V. N., *Introduction to Minimax*, John Wiley & Sons, 1974.

4. Graupe, D., *Identification of Systems*, Krieger, 1972.

5. Graupe, D., *Time Series Analysis: Identification and Adaptive Filtering*, Krieger, 1983.

6. Ruckdeschel, F. F., *Basic Scientific Subroutines*, Vol. 2, BYTE Publications, Inc., 1981.

## Suggested Additional Reading

1. Rivlin, T., *An Introduction to the Approximation of Functions*, Dover, 1969.

2. Davis, P. J., *Interpolation and Approximation*, Dover, 1975.

## Appendix: Derivation of Least Squares Algorithms

Much of the material that follows is based on the literature. Readers may consult the references accompanying this article as a starting place for additional material on the subject. I use nomenclature from the text.

### Batch Least Squares

Let MSE ≡ Sample Mean Squared Error

then $MSE = 1/k \sum_{i=1}^{k} e_i^2$

or $MSE = 1/k\ (\underline{e}^T\underline{e})$

Let tr ≡ trace operator ≡ sum of diagonal elements of a square matrix.

since $\underline{e}^T\underline{e} = tr(\underline{e}\underline{e}^T)$

and $\underline{e} = \underline{y} - X\underline{a}$

then $MSE = 1/k\ tr[(\underline{y} - X\underline{a})\ (\underline{y} - X\underline{a})^T]$

$$MSE = 1/k\ tr[(\underline{y} - X\underline{a})(\underline{y}^T - \underline{a}^T X^T)]$$

$$MSE = 1/k\ tr[\underline{y}\underline{y}^T + X\underline{a}\underline{a}^T X^T - X\underline{a}\underline{y}^T - \underline{y}\underline{a}^T X^T]$$

$$MSE = 1/k\,[tr(\underline{y}\underline{y}^T) + tr(X\underline{a}\underline{a}^T X^T) - tr(X\underline{a}\underline{y}^T) - tr(\underline{y}\underline{a}^T X^T)]$$

The gradient of a scaler, c, with respect to a matrix, Z, is defined as:

$$\frac{\partial c}{\partial Z} \equiv \begin{bmatrix} \frac{\partial c}{\partial z_{11}} & \frac{\partial c}{\partial z_{12}} & \cdots & \frac{\partial c}{\partial z_{1n}} \\ \frac{\partial c}{\partial z_{21}} & \frac{\partial c}{\partial z_{22}} & \cdots & \frac{\partial c}{\partial z_{2n}} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \frac{\partial c}{\partial z_{m1}} & \frac{\partial c}{\partial z_{m2}} & \cdots & \frac{\partial c}{\partial z_{mn}} \end{bmatrix}$$

Graupe [4] gives rules for the gradation of trace functions. Those of relevance here are:

$$\frac{\partial}{\partial Z}tr(A) = [0]$$

$$\frac{\partial}{\partial Z}tr(AZZ^TB) = (A^TB^T + BA)Z$$

$$\frac{\partial}{\partial Z}tr(AZB) = A^TB^T$$

Since the MSE is a positive hyperparabolic function of $\underline{a}$, a sufficient condition for minimizing the MSE is:

$$\frac{\partial MSE}{\partial a_i} = 0 \text{ for all } i$$

$$\frac{\partial}{\partial \underline{a}} MSE = \underline{0}$$

therefore

$$\frac{\partial}{\partial \underline{a}}tr(\underline{y}\underline{y}^T) + \frac{\partial}{\partial \underline{a}}tr(X\underline{a}\underline{a}^TX^T) - \frac{\partial}{\partial \underline{a}}tr(X\underline{a}\underline{y}^T) - \frac{\partial}{\partial \underline{a}}tr(\underline{y}\underline{a}^TX^T) = \underline{0}$$

Applying the rules for gradation of trace functions:

$$\frac{\partial}{\partial \underline{a}}tr(\underline{y}\underline{y}^T) = \underline{0}$$

$$\frac{\partial}{\partial \underline{a}}tr(X\underline{a}\underline{a}^TX^T) = 2X^TX\underline{a}$$

$$\frac{\partial}{\partial \underline{a}}tr(X\underline{a}\underline{y}^T) = X^T\underline{y}$$

$$\frac{\partial}{\partial \underline{a}}tr(\underline{y}\underline{a}^TX^T) = \frac{\partial}{\partial \underline{a}}tr(X\underline{a}\underline{y}^T) = X^T\underline{y}$$

thus

$$\underline{0} + 2X^TX\underline{a} - X^T\underline{y} - X^T\underline{y} = \underline{0}$$

$$X^TX\underline{a} - X^T\underline{y} = \underline{0}$$

$$\underline{a} = (X^TX)^{-1}X^T\underline{y}$$

### Sequential Least Squares

This algorithm provides an efficient means of performing regression on a growing data set. It does this by directly updating Q and $\underline{a}$, one data point at a time, without actually recording the data. However, to show mathematical equivalency to the Batch Least Squares algorithm above, we must imagine a growing $\underline{y}$, and X:

$$\underline{y}_k = \begin{bmatrix} \underline{y}_{k-1} \\ y_k \end{bmatrix} \text{ and } X_k = \begin{bmatrix} X_{k-1} \\ \underline{x}_k^T \end{bmatrix}$$

Observe that $\underline{y}_{k-1}$ and $X_{k-1}$ represent old data and $(\underline{x}_k, y_k)$ is a new data point that is being added to form the new data set represented by $\underline{y}_k$ and $X_k$. Then starting with the previously derived Batch Least Squares:

$$\underline{a}_k = (X_k^TX_k)^{-1}X_k^T\underline{y}_k$$

$$Q_k = (X_k^TX_k)^{-1}$$

$$Q_k^{-1} = X_k^TX_k$$

$$Q_k^{-1} = \sum_{i=1}^{k} \underline{x}_i\underline{x}_i^T$$

$$Q_k^{-1} = \sum_{i=1}^{k-1} \underline{x}_i\underline{x}_i^T + \underline{x}_k\underline{x}_k^T$$

$$Q_k^{-1} = Q_{k-1}^{-1} + \underline{x}_i\underline{x}_i^T$$

Next, the Matrix Inversion Lemma will be derived and used. Consider any nonsingular square matrices A and C and conformable matrices B and D:

$$D + DA^{-1}BCD = D + DA^{-1}BCD$$

$$DA^{-1}(A + BCD) = (C^{-1} + DA^{-1}B)CD$$

$$(C^{-1} + DA^{-1}B)^{-1}DA^{-1}(A + BCD) = CD$$

$$A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}(A + BCD) = A^{-1}BCD$$

$$I + A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}(A + BCD) = I + A^{-1}BCD$$

$$I + A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}(A + BCD) = A^{-1}(A + BCD)$$

$$(A + BCD)^{-1} + A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1} = A^{-1}$$

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}$$

Since $Q_k = (Q_{k-1}^{-1} + \underline{x}_k\underline{x}_k^T)^{-1}$

then by the Matrix Inversion Lemma:

$$Q_k = Q_{k-1} - Q_{k-1}\underline{x}_k(1 + \underline{x}_k^TQ_{k-1}\underline{x}_k)^{-1}\underline{x}_k^TQ_{k-1}$$

$$Q_k = Q_{k-1} - \frac{Q_{k-1}\,\underline{x}_k\,\underline{x}_k^T\,Q_{k-1}}{1 + \underline{x}_k^T\,Q_{k-1}\,\underline{x}_k}$$

$$\underline{a}_k = Q_kX_k^T\underline{y}_k$$

$$X_k^T\underline{y}_k = \sum_{i=1}^{k} \underline{x}_iy_i$$

$$X_k^T\underline{y}_k = \sum_{i=1}^{k-1} \underline{x}_iy_i + \underline{x}_ky_k$$

$$X_k^T\underline{y}_k = X_{k-1}^T\underline{y}_{k-1} + \underline{x}_ky_k$$

$$X_k^T\underline{y}_k = Q_{k-1}^{-1}Q_{k-1}X_{k-1}^T\underline{y}_{k-1} + \underline{x}_ky_k$$

$$\underline{a}_{k-1} = Q_{k-1}X_{k-1}^T\underline{y}_{k-1}$$

$$X_k^T\underline{y}_k = Q_{k-1}^{-1}\underline{a}_{k-1} + \underline{x}_ky_k$$

$$Q_{k-1}^{-1} = Q_k^{-1} - \underline{x}_k\underline{x}_k^T$$

$$X_k^T\underline{y}_k = (Q_k^{-1} - \underline{x}_k\underline{x}_k^T)\,\underline{a}_{k-1} + \underline{x}_ky_k$$

$$\underline{a}_k = Q_kX_k^T\underline{y}_k$$

$$\underline{a}_k = Q_k[(Q_k^{-1} - \underline{x}_k\underline{x}_k^T)\,\underline{a}_{k-1} + \underline{x}_ky_k]$$

$$\underline{a}_k = (Q_kQ_k^{-1} - Q_k\underline{x}_k\underline{x}_k^T)\,\underline{a}_{k-1} + Q_k\underline{x}_ky_k$$

$$\underline{a}_k = (I - Q_k\underline{x}_k\underline{x}_k^T)\,\underline{a}_{k-1} + Q_k\underline{x}_ky_k$$

$$\underline{a}_k = \underline{a}_{k-1} + Q_k\underline{x}_k\,(y_k - \underline{x}_k^T\underline{a}_{k-1})$$

If the Sequential Least Squares algorithm is to be used from a cold start, the initial values, $\underline{a}_0 = \underline{0}$, and Qo = cI should be used where c is the largest number possible without causing failure of the algorithm due to roundoff errors. Ideally, $Q_0^{-1}$ should equal [0] but, since this is singular, an ideal Qo does not exist. Hence, for any finite Qo, an error is introduced into the calculation. Making Qo large minimizes this error. There are, however, ways to completely avoid this error, which should make the SLS algorithm at least as accurate as algorithms such as Orthogonal Decomposition. But that is a topic for another article. ■■J

**(Listings begin on page 98)**

**Reader Ballot**
Vote for your favorite feature/article.
Circle Reader Service **No. 194.**

```
Minimax Fit
Function:  SIN(X*PI/2)      Range: 0.0 to 1.0
Maximum Absolute Relative
  Error = 1.08178979E-04
Error Variance = 5.82237977E-09
Coefficients:
A(0) =  1.5706264010388677E+00
A(1) = -6.4322566310483564E-01
A(2) =  7.2707440066142992E-02
```

$$\text{SIN}\left(\frac{\pi}{2}x\right) \approx a_0x + a_1x^3 + a_2x^5,\ -1 \leq x \leq 1$$

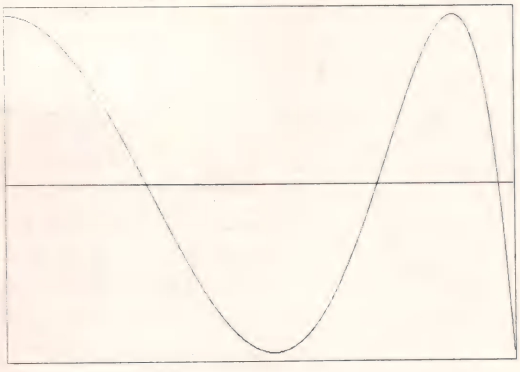


**Figure 1**

The Relative Error Curve Resulting from the Minimax Fitting of a 5th Degree (n=3) Polynomial to SIN(PI*X/2)

```
Least Squares Fit
Function:  SIN(X*PI/2)      Range: 0.0 to 1.0
Maximum Absolute Relative
  Error = 2.40870170E-04
Error Variance = 4.50629649E-09
Coefficients:
A(0) =  1.5706763023886539E+00
A(1) = -6.4370505986273774E-01
A(2) =  7.3269627643904808E-02
```

$$\text{SIN}\left(\frac{\pi}{2}x\right) \approx a_0x + a_1x^3 + a_2x^5,\ -1 \leq x \leq 1$$

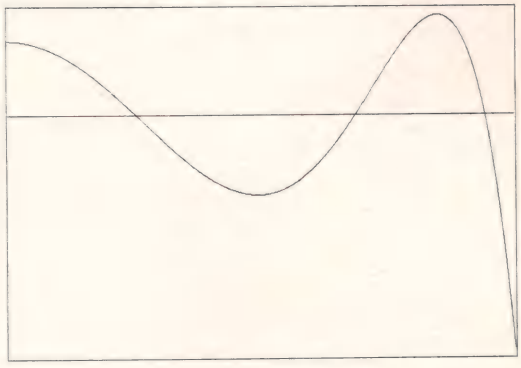


**Figure 2**

The Relative Error Curve Resulting from a Least Squares Fit of a 5th Degree (n=3) Polynomial to SIN(PI*X/2)

```
 Ruckdeschel:
Minimax Fit
Function:  SIN(X*PI/2)      Range: 0.0 to 1.0
Maximum Absolute Relative
  Error = 1.43752885E-04
Error Variance = 8.49330500E-09
Coefficients:
A(0) =  1.5706894000000000E+00
A(1) = -6.4323300000000005E-01
A(2) =  7.2555999999999996E-02
```

$$\text{SIN}\left(\frac{\pi}{2}x\right) \approx a_0x + a_1x^3 + a_2x^5,\ -1 \leq x \leq 1$$

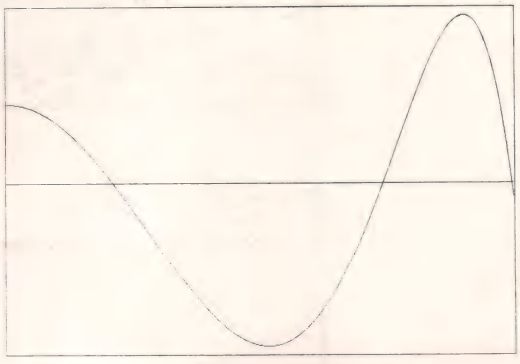


**Figure 3**

The Relative Error Curve Resulting from Ruckdeschel's Attempt to Achieve a Minimax Fit Using Optimization by Steepest Descent to SIN(PI*X/s)

**Figures 4 through 21**

Figures 4 through 21 represent approximations found by the minimax algorithm implemented in the Forth Matrix Language program of Listing 2. The even numbered figures duplicate the highest ordered approximations found by Hastings whereas the odd numbered figures represent approximations of a significantly higher order than those of Hastings.

```
Minimax Fit
Function: LOG(X)   Range: 1.0 to SQRT(10.0)
Maximum Absolute Error = 1.32254277E-07
Error Variance = 8.71559890E-15
Coefficients:
A(0) =  8.6859170033741495E-01
A(1) =  2.8933610841477914E-01
A(2) =  1.7751576585077944E-01
A(3) =  9.4403559884429847E-02
A(4) =  1.9129734604173343E-01
```

$$LOG_{10}(x) \approx a_0\left(\frac{x-1}{x+1}\right) + a_1\left(\frac{x-1}{x+1}\right)^3 + \cdots + a_4\left(\frac{x-1}{x+1}\right)^9,\ \frac{1}{\sqrt{10}} \leqslant x \leqslant \sqrt{10}$$

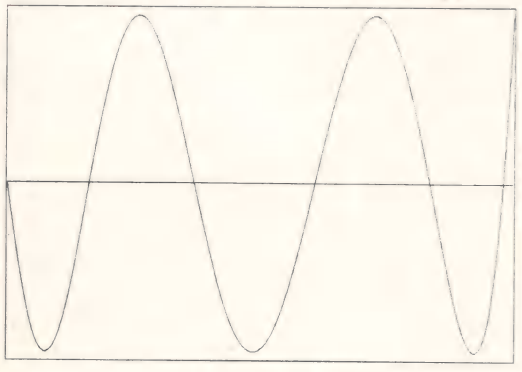


**Figure 4**

```
Minimax Fit
Function: LOG(X)   Range: 1.0 to SQRT(10.0)
Maximum Absolute Error = 1.11137499E-15
Error Variance = 5.76209886E-31
Coefficients:
A(0) =  8.6858896380645279E-01
A(1) =  2.8952965462124397E-01
A(2) =  1.7371779062864265E-01
A(3) =  1.2408424804953024E-01
A(4) =  9.6506696899528205E-02
A(5) =  7.9019310730381306E-02
A(6) =  6.6160376582026767E-02
A(7) =  6.2932880597822538E-02
A(8) =  2.5321022483988299E-02
A(9) =  1.3171606514518003E-01
A(10) = -1.3290449657183437E-01
A(11) =  2.1590497542362960E-01
```

$$LOG_{10}(x) \approx a_0\left(\frac{x-1}{x+1}\right) + a_1\left(\frac{x-1}{x+1}\right)^3 + \cdots + a_{11}\left(\frac{x-1}{x+1}\right)^{23},\ \frac{1}{\sqrt{10}} \leqslant x \leqslant \sqrt{10}$$

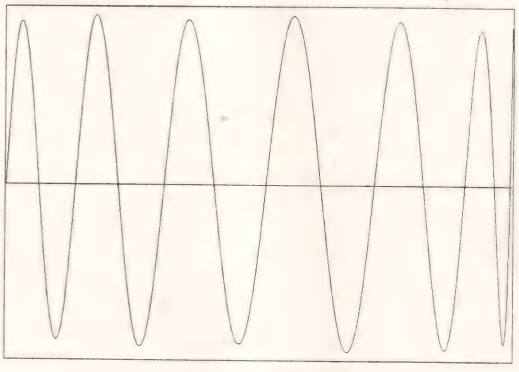


**Figure 5**

```
Minimax Fit
Function:  ARCTAN(X)   Range: 0.0 to 1.0
Maximum Absolute Error = 3.75119864E-08
Error Variance = 7.01558800E-16
Coefficients:
A(0) =  9.9999933573499511E-01
A(1) = -3.3329861463931842E-01
A(2) =  1.9946573483723884E-01
A(3) = -1.3908668702150279E-01
A(4) =  9.6422971343165645E-02
A(5) = -5.5913682046468495E-02
A(6) =  2.1863890550873954E-02
A(7) = -4.0548228214270449E-03
```

$$ARCTAN(x) \approx a_0x + a_1x^3 + \cdots + a_7x^{15},\ -1 \leqslant x \leqslant 1$$

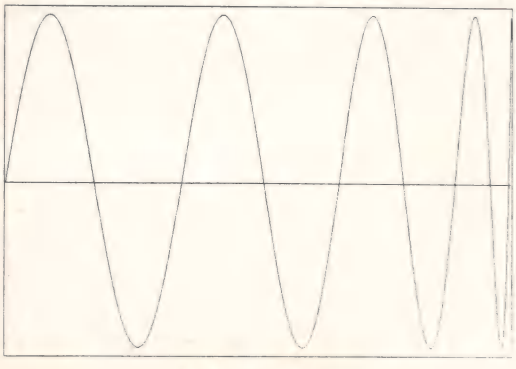


**Figure 6**

```
Minimax Fit
Function:  ARCTAN(X)   Range: 0.0 to 1.0
Maximum Absolute Error = 2.25113635E-11
Error Variance = 2.44028609E-22
Coefficients:
A(0) =  9.9999999943085005E-01
A(1) = -3.3333327048062561E-01
A(2) =  1.9999793768255469E-01
A(3) = -1.4282554529129779E-01
A(4) =  1.1083694502163881E-01
A(5) = -8.9412233101935801E-02
A(6) =  7.1433729243175675E-02
A(7) = -5.2520067625352863E-02
A(8) =  3.2239154415053844E-02
A(9) = -1.4726129937627916E-02
A(10) =  4.2957241092664187E-03
A(11) = -5.8808009033101821E-04
```

$$ARCTAN(x) \approx a_0x + a_1x^3 + \cdots + a_{11}x^{23},\ -1 \leqslant x \leqslant 1$$

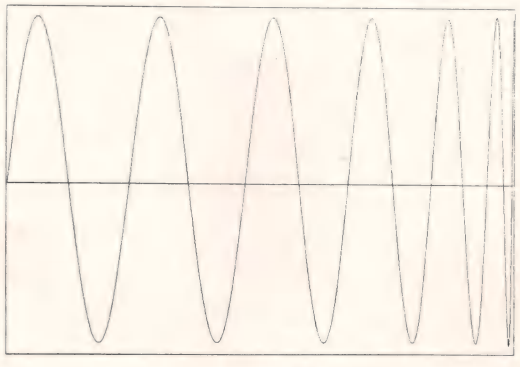


**Figure 7**

```
Minimax Fit
Function:  SIN(X*PI/2)      Range: 0.0 to 1.0
Maximum Absolute Relative
  Error = 5.31747856E-09
Error Variance = 1.41003464E-17
Coefficients:
A(0) =  1.5707963184491878E+00
A(1) = -6.4596371064188951E-01
A(2) =  7.9689679172657638E-02
A(3) = -4.6737669936775450E-03
A(4) =  1.5148532647329515E-04
```

$$\text{SIN}\left(\frac{\pi}{2}x\right) \approx a_0x + a_1x^3 + \cdots + a_4x^9,\ -1 \leqslant x \leqslant 1$$

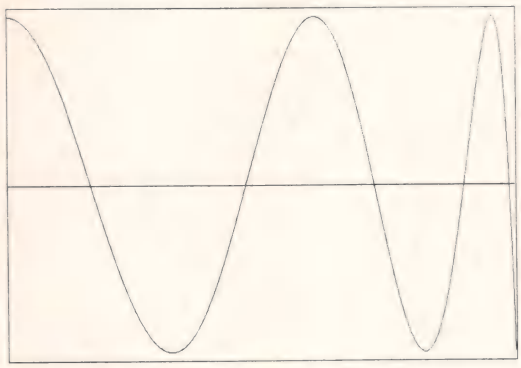


**Figure 8**

```
Minimax Fit
Function:  SIN(X*PI/2)      Range: 0.0 to 1.0
Maximum Absolute Relative
  Error = 6.28158853E-14
Error Variance = 1.94727887E-27
Coefficients:
A(0) =  1.5707963267947991E+00
A(1) = -6.4596409749718697E-01
A(2) =  7.9692626107052855E-02
A(3) = -4.6817533267793027E-03
A(4) =  1.6043893116473893E-04
A(5) = -3.5955991943598100E-06
A(6) =  5.4590205930680611E-08
```

$$\text{SIN}\left(\frac{\pi}{2}x\right) \approx a_0x + a_1x^3 + \cdots + a_6x^{13},\ -1 \leqslant x \leqslant 1$$

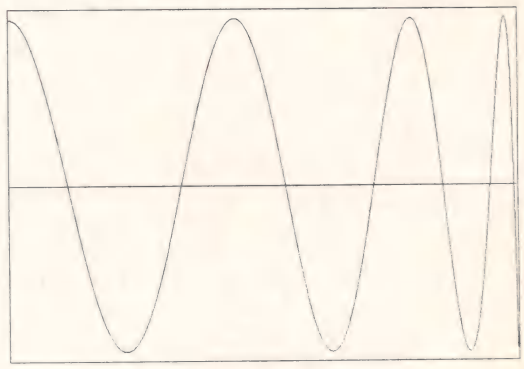


**Figure 9**

```
Minimax Fit
Function: 10^X Range: 0.0 to 1.0
Maximum Absolute Relative Error = 4.96196032E-09
Error Variance = 1.22996227E-17
Coefficients:
A(0) =  1.151292774666438 1E+00
A(1) =  6.6273090331202300E-01
A(2) =  2.5439346793918238E-01
A(3) =  7.2952042631990802E-02
A(4) =  1.7420655137681311E-02
A(5) =  2.5552719905680065E-03
A(6) =  9.3253665510615105E-04
```

$$10^x \approx (1 + a_0x + a_1x^2 + \cdots + a_6x^7)^2,\ 0 \leqslant x \leqslant 1$$

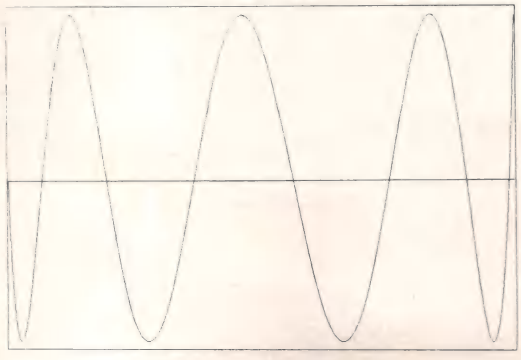


**Figure 10**

```
Minimax Fit
Function: 10^X Range: 0.0 to 1.0
Maximum Absolute Relative Error = 1.18194099E-13
Error Variance = 6.91798176E-27
Coefficients:
A(0) =  1.1512925464870498E+00
A(1) =  6.6273726433125657E-01
A(2) =  2.5433481471199015E-01
A(3) =  7.3203528479750382E-02
A(4) =  1.6855309537847786E-02
A(5) =  3.2355525426469244E-03
A(6) =  5.2954928235629073E-04
A(7) =  7.9479944032521686E-05
A(8) =  7.6291259328835485E-06
A(9) =  1.9857253301916773E-06
```

$$10^x \approx (1 + a_0x + a_1x^2 + \cdots + a_9x^{10})^2,\ 0 \leqslant x \leqslant 1$$

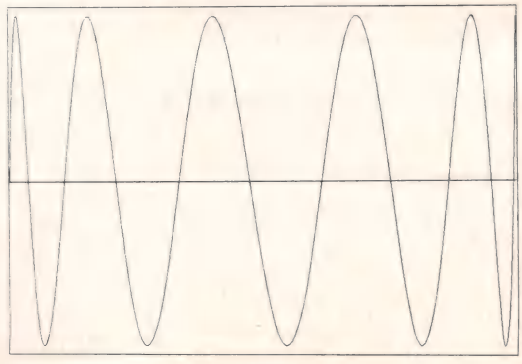


**Figure 11**

```
Minimax Fit
Function: (1/SQRT(2*PI))*EXP(-X*X/2)
  Range: 0.0 to 5.0
Maximum Absolute Error = 2.21755841E-04
Error Variance = 2.27665975E-08
Coefficients:
A(0) =  2.5052394568638987E+00
A(1) =  1.2830816024602694E+00
A(2) =  2.2655669890401081E-01
A(3) =  1.3058266628015777E-01
A(4) = -2.0229823620712177E-02
A(5) =  3.9113765898126695E-03
```

$$\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}} \approx \frac{1}{a_0 + a_1x^2 + a_2x^4 + \cdots + a_5x^{10}},\ -\infty<x<\infty$$

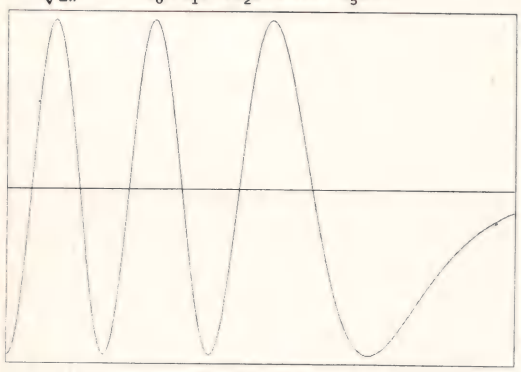


**Figure 12**

```
Minimax Fit
Function: (1/SQRT(2*PI))*EXP(-X*X/2)
   Range: 0.0 to 6.5
Maximum Absolute Error = 2.32244563E-09
Error Variance = 2.58681580E-18 Coefficients:
A(0) =  2.5066282606021772E+00
A(1) =  1.2533150158068533E+00
A(2) =  3.1332010186080966E-01
A(3) =  5.2251507235153920E-02
A(4) =  6.4745038039304478E-03
A(5) =  7.0651126222071438E-04
A(6) =  2.0618249722321060E-05
A(7) =  1.7799505117036926E-05
A(8) = -3.6392970226935947E-06
A(9) =  7.6125309644419998E-07
A(10) = -9.9471514152083657E-08
A(11) =  9.3097523153405433E-09
A(12) = -5.8145287978472377E-10
A(13) =  2.3753990217929335E-11
A(14) = -5.6523614575329404E-13
A(15) =  6.2370941540151992E-15
```

$$\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}} \approx \frac{1}{a_0 + a_1x^2 + a_2x^4 + \cdots + a_{15}x^{30}},\ -\infty<x<\infty$$

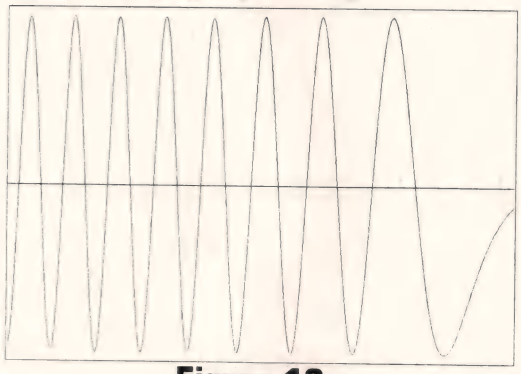


**Figure 13**

```
Minimax Fit
Function: (PI/2 - ARCSIN(X))/SQRT(1-X)
  Range: 0.0 to 1.0
Maximum Absolute Error = 2.18035876E-08
Error Variance = 2.36955457E-16
Coefficients:
A(0) =  1.5707963050017066E+00
A(1) = -2.1459880378710242E-01
A(2) =  8.8979074758192392E-02
A(3) = -5.0174698127643524E-02
A(4) =  3.0893003101670889E-02
A(5) = -1.7089738000951291E-02
A(6) =  6.6712412050773488E-03
A(7) = -1.2628162735097773E-03
```

$$\text{ARCSIN}(x) \approx \frac{\pi}{2} - \sqrt{1-x}\,(a_0 + a_1x + \cdots + a_7x^7),\ 0\leq x\leq 1$$

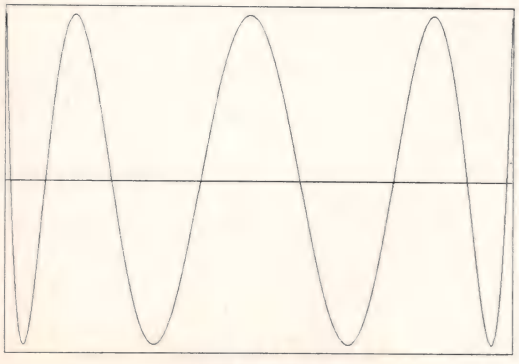


**Figure 14**

```
Minimax Fit
Function: (PI/2 - ARCSIN(X))/SQRT(1-X)
  Range: 0.0 to 1.0
Maximum Absolute Error = 1.60433862E-12
Error Variance = 1.27557003E-24
Coefficients:
A(0) =  1.5707963267933036E+00
A(1) = -2.1460183603317504E-01
A(2) =  8.9048588781845170E-02
A(3) = -5.0792024847596283E-02
A(4) =  3.3671623513562537E-02
A(5) = -2.4303150171847927E-02
A(6) =  1.8329760127140188E-02
A(7) = -1.3750906300991303E-02
A(8) =  9.5269440415934169E-03
A(9) = -5.5316843097328707E-03
A(10) =  2.4006330400566690E-03
A(11) = -6.6846646636149031E-04
A(12) =  8.7754386863194461E-05
```

$$\text{ARCSIN}(x) \approx \frac{\pi}{2} - \sqrt{1-x}\,(a_0 + a_1x + \cdots + a_{12}x^{12}),\ 0\leq x\leq 1$$

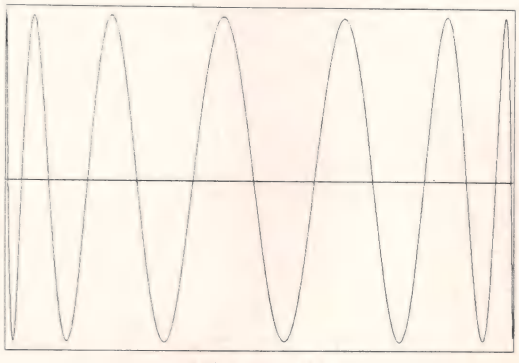


**Figure 15**

```
Minimax Fit
Function:  LN(1+X)      Range: 0.0 to 1.0
Maximum Absolute Error = 3.21271115E-08
Error Variance = 5.14899243E-16
Coefficients:
A(0) =  9.9999642149771728E-01
A(1) = -4.9987400637711582E-01
A(2) =  3.3179798745075179E-01
A(3) = -2.4072984401903777E-01
A(4) =  1.6764614981179890E-01
A(5) = -9.5320711256370566E-02
A(6) =  3.6083546727710289E-02
A(7) = -6.4523953561366849E-03
```

$$LN(1+x) \approx a_0x + a_1x^2 + \cdots + a_7x^8,\ 0 \leqslant x \leqslant 1$$

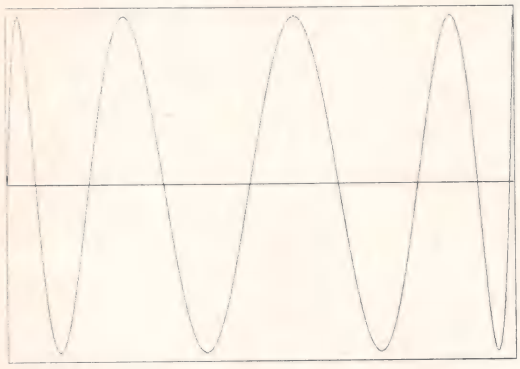


**Figure 16**

```
Minimax Fit
Function:  LN(1+X)      Range: 0.0 to 1.0
Maximum Absolute Error = 2.99005943E-12
Error Variance = 4.44361184E-24
Coefficients:
A(0) =  9.9999999921292859E-01
A(1) = -4.9999993436563822E-01
A(2) =  3.3333141255453208E-01
A(3) = -2.4997150728221187E-01
A(4) =  1.9974825550112713E-01
A(5) = -1.6521637838885567E-01
A(6) =  1.3708823785361379E-01
A(7) = -1.0851242464256000E-01
A(8) =  7.6176582881979454E-02
A(9) = -4.3423490764673826E-02
A(10) =  1.8141724720925251E-02
A(11) = -4.8159341987676569E-03
A(12) =  6.0063748040789543E-04
```

$$LN(1+x) \approx a_0x + a_1x^2 + \cdots a_{12}x^{13},\ 0 \leqslant x \leqslant 1$$

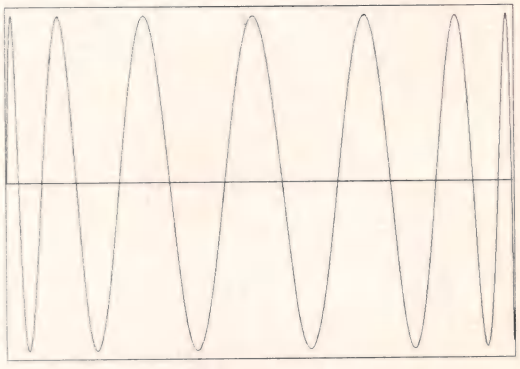


**Figure 17**

```
Minimax Fit
Function: EXP(-X) Range: 0.0 to 16.0
Maximum Absolute Error = 2.42142191E-07
Error Variance = 2.61153603E-14
Coefficients:
A(0) =  2.4999867539512141E-01
A(1) =  3.1257613051553959E-02
A(2) =  2.5913379887911865E-03
A(3) =  1.7157681582990019E-04
A(4) =  5.4274220448181193E-06
A(5) =  6.9077386115360822E-07
```

$$e^{-x} \approx \frac{1}{(1 + a_0x + a_1x^2 + \cdots + a_5x^6)^4},\ 0 \leqslant x < \infty$$

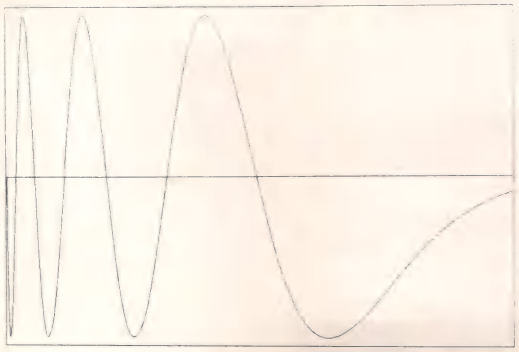


**Figure 18**

```
Minimax Fit
Function: EXP(-X) Range: 0.0 to 20.0
Maximum Absolute Error = 3.42649682E-13
Error Variance = 6.04906522E-26
Coefficients:
A(0) =  2.4999999999648442E-01
A(1) =  3.1250000041451426E-02
A(2) =  2.6041665079503278E-03
A(3) =  1.6276070164139176E-04
A(4) =  8.1377395547909460E-06
A(5) =  3.3925041887012053E-07
A(6) =  1.2048462387673388E-08
A(7) =  3.9315765909007019E-10
A(8) =  8.2512108449021085E-12
A(9) =  4.8158149678826166E-13
A(10) = -6.5346922729955708E-15
A(11) =  4.8211465914063989E-16
```

$$e^{-x} \approx \frac{1}{(1 + a_0x + a_1x^2 + \cdots + a_{11}x^{12})^4},\ 0 \leqslant x < \infty$$

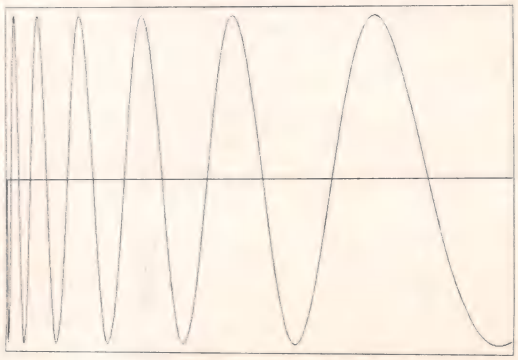


**Figure 19**

```
Minimax Fit
Function: 1-ERF(X) Range: 0.0 to 4.0
Maximum Absolute Error = 2.59522537E-07
Error Variance = 2.81576631E-14
Coefficients:
A(0) =  7.0523079991967080E-02
A(1) =  4.2281992401241905E-02
A(2) =  9.2705995340344790E-03
A(3) =  1.5190656505008346E-04
A(4) =  2.7663627971689751E-04
A(5) =  4.3048078931175112E-05
```

$$ERF(x) = 1 - \frac{1}{(1 + a_0x + a_1x^2 + \cdots + a_5x^6)^{16}},\ 0 \leqslant x \leqslant \infty$$

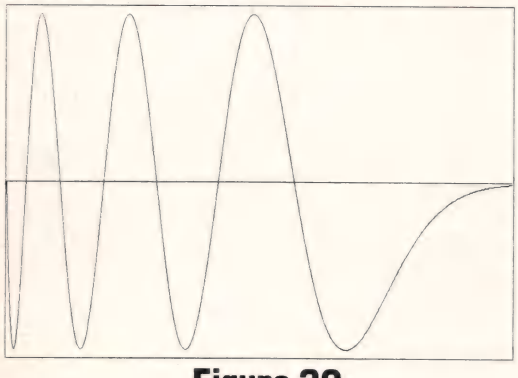


**Figure 20**

```
Minimax Fit
Function: 1-ERF(X) Range: 0.0 to 4.2
Maximum Absolute Error = 1.41490525E-12
Error Variance = 1.02491312E-24
Coefficients:
A(0) =  7.0523697931302184E-02
A(1) =  4.2275532274750102E-02
A(2) =  9.2877883593596861E-03
A(3) =  1.4906829666204733E-04
A(4) =  2.3210702049655493E-04
A(5) =  1.0664165183504473E-04
A(6) = -3.4002731629183212E-05
A(7) =  4.8035866639544990E-06
A(8) =  2.8758813985644038E-06
A(9) = -1.8271251310999870E-06
A(10) =  5.5554105470722536E-07
A(11) = -1.0046197131329913E-07
A(12) =  1.0438113302017026E-08
A(13) = -4.7894501157677281E-10
```

$$ERF(x) = 1 - \frac{1}{(1 + a_0x + a_1x^2 + \cdots + a_{13}x^{14})^{16}},\ 0 \leqslant x \leqslant \infty$$

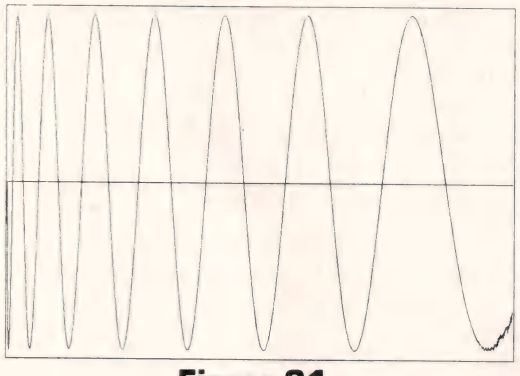


**Figure 21**

# Minimax (Text begins on page 84)

## Listing One

```
100 '   MINIMAX VIA ERROR FLUFFING ALGORITHM by Steven A. Ruzinsky
      This program demonstrates the determination of polynomial
      coefficients to a minimum maximum absolute error criterion. The result
110 ' is better than that of Hastings, Approximations for Digital Computers,
      1955, Princeton University Press, p 138.  This program is in IBM Basic.
120 ' ------------------------------------------------------------------------

130 DEFDBL A-Z : DEFINT I-N
140 N = 3 : M = 50 : ITERATIONS = 1000
150 DIM X(M,N) , Y(M) , XK(N) , Q(N,N) , QX(N) , AK(N) , A(N)
160 ' ------------------------------------------------------------------------
170 ' The following fills matrices Y(i) and X(i,j) with data.  The function used
      is SIN(PI*X/2), however, the data is modified so that the minimax
      criterion is applied to the relative error :
180 FOR I = 1 TO M
190 Y(I) = 1
200 E = I/M
210 D = 1/SIN(1.570796327#*E)
220 FOR J = 1 TO N
230 X(I,J) = D*E^(J+J-1)
240 NEXT J , I
250 ' ------------------------------------------------------------------------
260 ' The following initiates the Q matrix.  It may be necessary to
      to adjust the number in line 280 for best results.
270 FOR I = 1 TO N
280 Q(I,I) = 1000000!
290 NEXT I
300 ' ------------------------------------------------------------------------
310 ' The following loop with index, K, reiterates the sequential least squares
      algorithm.  Up to limit M, each data point is incorporated once into Q and
      AK. This results in a least squares fit to the data. Afterwards, the data
320 ' corresponding to the  maximum absolute error are reincorporated back into
      Q and AK.
330 EBEST = 1
340 FOR K = 1 TO ITERATIONS + M
350 IF K > M THEN GOSUB 740 ELSE GOSUB 650
360 D = 1
370 FOR J = 1 TO N
380 QX = 0
390 FOR I = 1 TO N
400 QX = QX + XK(I)*Q(J,I)
410 NEXT I
420 QX(J) = QX
430 D = D + XK(J)*QX
440 NEXT J
450 FOR J = 1 TO N
460 QX = QX(J)/D
470 FOR I = 1 TO N
480 Q(I,J) = Q(I,J) - QX(I)*QX
490 NEXT I , J
500 FOR J = 1 TO N
510 QX = 0
520 FOR I = 1 TO N
530 QX = QX + XK(I)*Q(J,I)
540 NEXT I
550 AK(J) = AK(J) + QX*E
560 NEXT J , K
570 ' ------------------------------------------------------------------------
580 ' The following prints the results :
590 PRINT "Coefficients:" : FOR I= 1 TO N
```

```
600 AK(I) = A(I)
610 PRINT AK(I)
620 NEXT I : GOSUB 740 : END
630 ' ------------------------------------------------------------------------
640 ' Subroutine for incorporating each data point once :
650 E = Y(K)
660 FOR I = 1 TO N
670 XK = X(K,I)
680 XK(I) = XK
690 E = E - AK(I)*XK
700 NEXT I
710 RETURN
720 ' ------------------------------------------------------------------------
730 ' Subroutine for finding maximum absolute error and corresponding data
      point :
740 EMAX = 0 : JMAX = 1
750 FOR J = 1 TO M
760 E = Y(J)
770 FOR I = 1 TO N
780 E = E - X(J,I)*AK(I)
790 NEXT I
800 E = ABS(E)
810 IF E > EMAX THEN EMAX = E : JMAX = J
820 NEXT J
830 PRINT "Iterations =", K-M, "Maximum Absolute Error =", EMAX
840 E = Y(JMAX)
850 FOR I = 1 TO N
860 XK = X(JMAX,I)
870 XK(I) = XK
880 E = E - AK(I)*XK
890 NEXT I
900 IF EMAX < EBEST THEN EBEST = EMAX : GOTO 940
910 RETURN
920 ' ------------------------------------------------------------------------
930 ' Subroutine for saving best coefficients, A :
940 FOR I=1 TO N
950 A(I) = AK(I)
960 NEXT I
970 RETURN
```

End Listing One

## Listing Two

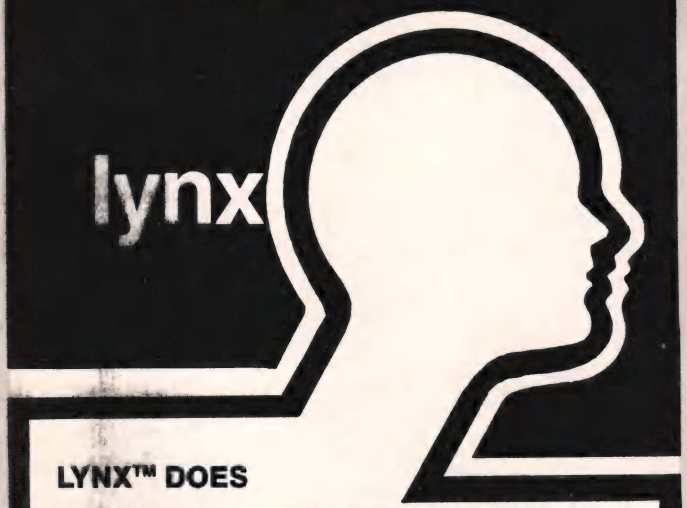
```
201 LIST

 0 ( ------------ Minimax Algorithm, screen 1 of 7 ---------------)
 1 (      Minimax Via Error Fluffing by Steven A. Ruzinsky        )
 2 (  This program is written in Polyforth with FML Version 3.0   )
 3
 4 ( ------------ Matrix and Variable Definitions ----------------)
 5
 6 ( Enter range of x here : )       0.0 1.0
 7
 8 ( Enter dimension of X here : )   320 12
 9
10 2DUP matrices X x
11 LCONSTANT X2          LCONSTANT X1          0.0 LCONSTANT ek
12 CONSTANT n            CONSTANT m              0 CONSTANT k
13 0. 2CONSTANT i        1. 2CONSTANT 1.
14 m vectors y e         n vectors a A          n vectors xk CC
15 n vectors Qx xQ       xQ trnV              n n matrices Q QxxtQ
```

```
202 LIST

 0 ( ------------ Minimax Algorithm, screen 2 of 7 ---------------)
 1 : FUNCTION       ATAN ;
 2 : POLYNOMIAL     a CC **plyV ;
 3 : ERROR          FDUP FUNCTION FSWAP POLYNOMIAL F- ;
 4 : INITIALIZE     ( X x y e A a xk Qx Q QxxtQ CC )clrA
 5                  1.0 ['] ek L! 0. ['] i 2! 0 ['] k ! ;
 6 : INSERT-DATA    X1 X2 y ['] FUNCTION yA
 7                  X1 X2 e xA
 8                  CC indA
 9                  CC f2* (A)
10                  1.0 CC f+ A.S
11                  e CC X f** .X. ;
12
13 : e-CALC         X a e *A+ y e f- A.A ;
14 : E-CALC         e abs max red(A) ;
15 : V-CALC         e sqr f+ red(A) m >N F/ ;
```

*(Continued on page 101)*

# LYNX

the professional's replacement for Microsoft L80




**LYNX™ DOES EVERYTHING L80™ DOES AND MORE!**

**LYNX™ LETS YOU USE ALL AVAILABLE MEMORY.**

**You can create COM files that are 10K larger—without overlays—by replacing L80 with LYNX.**

**LYNX™ IS A FULL OVERLAY LINKER**

**Running twice as fast as its nearest competitor, LYNX is tree structured, multi-segmented and multi-leveled, with automatic or explicit overlay invocation. You can run programs that are larger than available memory.**

**LYNX™ HAS BEEN HELPING MICROSOFT FORTRAN PROGRAMMERS FOR YEARS—NOW IT IS ALSO AVAILABLE FOR MICROSOFT BASIC AND AZTEC C.™**

**LYNX™ IS A QUALITY PRODUCT from the same company who offers you:**

- **GrafTalk, the business graphics package for Micros**
- **GRAPHICS development tools**
- **2780/3780, 3270, X.25 communications**
- **MICRO TO MICRO communications**

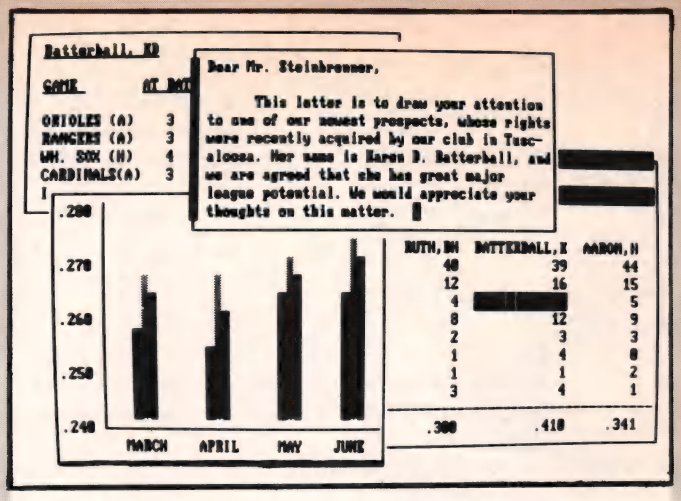
REDDING GROUP INC.

2730 High Ridge Road
Stamford, CT 06903
(203) 329-8874/Telex. 643351

**$250**

LYNX and GrafTalk are trademarks of Redding Group, Inc. L80 is a trademark of Microsoft. AZTEC C is a trademark of MANX Software Systems.

Circle **no. 60** on reader service card.




# SOFTWARE FUSION™

**With APX Core Executive™**

**The software that allows you to run as many as 8 programs simultaneously on your IBM PC.**

With APX Core™ you can coordinate off-the-shelf programs through overlapping windows and optimize keystrokes to reduce repetitive tasks to a single command. APX Core™ even lets you "cut and paste" data among different programs.

**$95 special introductory offer**

APX

**Application Executive Corporation**

600 Broadway
Suite 4C
New York, NY 10012
(212) 226-6347

Circle **no. 4** on reader service card.

# A Software Bridge


**DOWNLOADING SERVICE**

- Port-A-Soft provides the service of taking programs from a diskette that a customer's computer cannot read and transferring it to a diskette that the customer's computer can read.
- Service available for approximately 250 diskette and tape formats from over 13 micro, mini, and main-frame operating systems.
- Disk to disk, tape to disk , disk to tape conversions.
- Fast service. One day disk conversions. 72-hour tape conversions.
- Competitive prices. Disk conversions as little as $5.00 per disk plus setup, shipping and handling.

**DOWNLOADING SOFTWARE**

- Port-A-Soft sells programs that make it possible for the customer's computer to read diskettes for many other computer makes and models.

**DOWNLOADING HARDWARE**

- Port-A-Soft sells specially designed computers and peripherals that support the reading, writing, and formatting of diskettes for many computer makes and models.

**WHO CAN BENEFIT?**

**USERS:** Enhance the power of your micro with programs not available in your diskette format, or with data such as mailing lists, taken from 9 track tapes.

**MANUFACTURERS AND DEALERS:** Let us help you make that sale that is conditioned on converting the customers data to the new computer, or let us help you provide the sale clinching software that the customer needs to opt for your product.

**SOFTWARE PUBLISHERS:** Expand your profits by letting us download your software to those unusual formats you cannot afford to support directly, or by porting your software to other operating systems and new markets.

**USERS GROUPS:** Get the public domain software you want in the format you want for your users group.

**PORT-A-SOFT**

423 E. 800 N. Orem, Utah 84057 (801) 226-6704

Circle **no. 49** on reader service card.

# Minimax (Listing Continued, text begins on page 84)

## Listing Two

```
203 LIST

 0 ( ------------ Minimax Algorithm, screen 3 of 7 ---------------)
 1
 2 : ZERO       FDROP 0.0 ;
 3 : PLOT-ZERO  oldSCALE 0.0 1.0 @x ['] ZERO yA @x pltA newSCALE ;
 4 : PLOT-e     MRES dark
 5              BYline .95 SCALE e pltA PLOT-ZERO 1.0 SCALE ;
 6 : HPLOT-e    HRES dark BYline .95 SCALE X1 X2 ['] ERROR PLOT
 7              PLOT-ZERO FRAME 1.0 SCALE MRES BELL ;
 8 : BATCH-LEAST-SQUARES  X @ T!T   @ 1/A   X y a regA  ;
 9
10 : RESULTS    HPLOT-e 27 EMIT 69 EMIT
11 CR 0= IF ." Least Squares Fit" ELSE ." Minimax Fit" THEN
12 CR ." Function: ARCTAN(X)     " ." Range: 0.0 to 1.0"
13 CR ." Maximum Absolute Error =" 2TOP abs max red(A) 8 S.
14 CR ." Error Variance =" 2TOP sqr f+ red(A) 2TOP elm >N F/ 8 S.
15 CR ." Coefficients:" 16 SCI a  V. 16 FIX CR ;


204 LIST

 0 ( ------------ Minimax Algorithm, screen 4 of 7 ---------------)
 1
 2 : NORMALIZE-@    m >N  n >N 10.0 F!  F/  @ f!  A.S ;
 3
 4 : REDUCE-e  e-CALC
 5             a @x A>A V-CALC ['] ek L! 50 0 DO
 6             e-CALC
 7             X e A rregA
 8             A a f+ A.A
 9             V-CALC CR FDUP I . 16 S. ek 16 S.
10             FDUP ek F< IF ['] ek L! a @x A>A  ELSE FDROP THEN
11             LOOP
12             1.0 ['] ek L!
13             @x a A>A  ;
14
15 : x>X       m 0 DO I x I X R>R LOOP ;


205 LIST

 0 ( ------------ Minimax Algorithm, screen 5 of 7 ---------------)
 1
 2 : SPEEDUP      I' = IF m 2/ ['] m !   X A e !A+   y e f- A.A
 3                e abs (A)   e x V>C   x X srtsC   e y srtsA
 4                y revC   X revC   X x A>A   m n X !dim
 5                m 1 y !dim   m 1 e !dim   x>X  m n x !dim THEN ;
 6
 7 : FIND-MAX(e),xk;RECORD-BEST-a,ek
 8                                  e-CALC
 9                                  e amax(i)
10                                  ['] k ! FDUP
11                                  ek F< IF
12                                  FDUP ['] ek L!
13                                  a A A>A
14                                  DROP 0 THEN
15                                  k X 0 xk R>C ;


206 LIST

 0 ( ------------ Minimax Algorithm, screen 6 of 7 ---------------)
 1
 2
 3 : SEQUENTIAL-LEAST-SQUARES
 4                           xk @ x@ !T+
 5                           @ xk @x !A+
 6                           @x x@ @xxt@ f! .X.
 7                           xk @x !V+
 8                           1.0 F+ 1/N @xxt@ f! A.S
 9                           @xxt@ @ -f A.A
10                           @ xk @x !A+
11                           k e V@
12                           @x f! A.S
13                           @x a f+ A.A  ;
14
15


207 LIST

 0 ( ------------ Minimax Algorithm, screen 7 of 7 ---------------)
 1 : MINIMAX
 2             INITIALIZE
 3             INSERT-DATA
 4             BATCH-LEAST-SQUARES
 5             REDUCE-E  PAGE 0 RESULTS  BELL KEY DROP
 6             NORMALIZE-@
 7             0 50 0 DO 1000 0 DO  i  1. D+ ['] i 2!
 8             FIND-MAX(e),xk;RECORD-BEST-a,ek
 9             CR i 7 D.R  k 10 U.R DUP 7 U.R 5 SPACES 8 S.
10             SEQUENTIAL-LEAST-SQUARES
11             1+ LOOP
12             0 SPEEDUP 5 SPEEDUP
13             DARK PLOT-e BELL BELL BELL LOOP
14             A a A>A
15             PAGE 1 RESULTS ;
```

**End Listings**

---

For those interested in FML, version 3.0 is available at a cost of $150 ppd exclusively from United Applied Research, Inc., P.O. Box 1164, North Riverside, IL 60546, and requires the IBM PC with an 8087 and at least 128K bytes of RAM (320K bytes preferred) and Polyforth with 8087 support.

---

# Languages and Parentheses

## A Suggestion for Forth-like Languages

by Ernest E. Bergman

Why do most languages use parentheses, brackets, or braces? In this article, we shall consider this question (and even discuss a language that doesn't use them!) by tracing an informal route from the philosophy of language and communication to the practicalities of compiler construction.

This article will also serve as something of a followup to the brief description of the public domain Forth-like language PISTOL which appeared in *DDJ* in February 1983. PISTOL will be used as an example in the discussion; both its virtue and its vice is a lack of parentheses. The discussion should also be applicable to some degree, however, to most languages which use a Reverse Polish syntax.

### Languages

Languages are for communicating. We use English to talk to each other. Fortran, developed as an aid to give computers instructions on how to carry out numerical calculations, was an early attempt at designing a language. Only a moderate success, Fortran's lack of regularity in syntax makes the design of the compiler's parser quite arduous; at the same time the programmer finds it difficult to remember the details of the syntax. BASIC then was developed, its syntax making it easier to learn and easier to parse.

A more recently developed language, Pascal has earned high marks for readability by both people and machines. The regularity of its syntax makes compiling easier and its readability by humans allows easier modification of the program ("maintenance"); that is, if you can understand the program, you are more likely to find its bugs and fix them, but if a program is incomprehensible, you will feel some inhibitions about expanding upon it.

Readability also means that a language can be a vehicle for interpersonal communications. (New algorithms are frequently described in Pascal.) Expressiveness is another important aspect of a language. It is easier to think of approaches to problems if the language supports useful concepts.

A "good" language, therefore, should be easy to write and capable of expressing the concepts needed for the implementation of algorithms. It should be easily read by people as well as by machines. We may not have a good language yet, but the present languages may yield fruitful approaches for better languages in the future.

### Parentheses in Algebraic Languages

For numerical calculations, we often need to provide a formula, such as:

```
VOLUME=LENGTH*WIDTH*
       HEIGHT
```

For more complicated formulas expressed in this algebraic format, we must resort to using parentheses:

```
TOTAL=(L1*W1*H1)+(L2*W2*H2)
```

Since many algebraic languages, such as Fortran, BASIC, and Pascal, assign a higher precedence to multiplication than to addition, the parentheses here are optional as far as the computer is concerned. But for people to read this, the parentheses are helpful (even essential) for comprehension. If the formula requires that the calculation be carried out in an order different from the normal precedence, the parentheses become mandatory. Additional parentheses, in addition to the mandatory pairs, are acceptable and may improve readability.

Because of the increased readability for humans, parentheses are frequently used in the design of computer languages. To write in a language that uses parentheses requires considerable training. For example, people accustomed to a Hewlett-Packard calculator, which uses RPN (Reverse Polish Notation), would run into problems using an algebraic calculator, such as those produced by Texas Instruments. While many people write in parentheses pairs after the rest of the equation format is written out, many users of HP calculators would find the RPN hard to understand if it were written out.

In summary, algebraic notation is easier to read and harder to write, while RPN, which does not use parentheses, is easier to write (if it is written sequentially) but harder to read (a somewhat random-access process). To parse an expression serially requires more work for a compiler if algebraic notation is used than if RPN is used.

### LISP

The use of parentheses in LISP (sometimes taken to stand for Lots of Irritating Single Parentheses) is very constrained. In list notation, parentheses are required but you may not add extra pairs optionally without altering the meaning. For example, the list A, B, C would be written (A B C), whereas the meaning of ((A B C)) is "the list containing the single element (A B C)"; it is a list of one list.

It is an interesting question whether an RPN version of LISP would be possible through constructions such as A B C LIST and A B C LIST LIST for (ABC) and ((ABC)), respectively. In addition to the question of ambiguities that might exist, we would have to assess the readability of the notation.

### PISTOL

PISTOL was developed to make a Forth-like language available to users

*by Ernest E. Bergmann Physics, Building 16, Lehigh University, Bethlehem, PA 18015.*

on large mainframe computers. Because it has been written both in Pascal and in C, it should be available with little conversion effort on many other computers.

The development of PISTOL involved certain conscious departures from Forth, including the adoption of some of the methods used in a similar language, STOIC (Stack-Oriented Incremental Compiler), which was developed as an extension to Forth. It is a shame that STOIC has not received as much attention as Forth since it is well conceived and documented. The most noticeable difference between Forth and STOIC or PISTOL is the latter's use of string literals to regularize the syntax further toward RPN and to increase the ease of handling strings.

In any case, all three languages (Forth, STOIC, and PISTOL) are largely precedence free and use an RPN style for expressing computations. Any reader who uses an HP calculator will find the description of these languages, which do not rely on paired parentheses to control precedence, quite familiar, although programming an HP calculator usually is not done in writing. Suppose we wish to evaluate:

$$(5+7)*13-7$$

We would type (for STOIC, PISTOL, and Forth):

```
5 7 + 1 3 * 7 - =
```

No parentheses are used. You would read this as: "take 5 and 7; add them; take 13; multiply it by the previous result; take 7; subtract it from the previous result; display the answer."

RPN notation is easier to write because no "special cases" occur with regard to precedence. If you have a "function" that takes three arguments and returns two results, it "fits" into the notation without any adjustments (and still without the need for parentheses). Variable numbers of arguments and results can be handled as well; in contrast, the Pascal function WRITELN takes a variable number of arguments and must, consequently, be a built-in feature of the compiler.

### Parsing and Code Generation

These days parsing methods readily applicable to algebraic expressions are well known. More often than not an intermediate code is produced, such as P-code. This code frequently is based upon some virtual stack machine.

Compiling expressions in a Forth-like language is extremely simple; you create code that pushes each literal onto a parameter stack as it is encountered in parsing the source code. Almost all other objects encountered in the parsing process simply require code that invokes the named function or procedure. The simplicity of the language reduces the programmer's work in writing source code: you do not have to remember the precedence of various operators since you are writing "directly" to the stack machine.

### Program Flow in Structured Languages

Almost all software must exercise a conditional transfer of program control, in addition to purely sequential execution. For example, in the C language the general two-way conditional branch is:

```
if (expression)
{true actions}
else
{false actions}
```

If the original expression, enclosed in parentheses, evaluates to "true," only the true actions, enclosed in the first set of braces, are carried out. Otherwise, only the false actions, enclosed in the second pair of braces, are used. The "else" portion is optional. If only a simple statement is used for the true or false action, the enclosing braces become optional.

The analogous structure in Pascal, which is also subject to analogous simplifications, is:

```
IF boolean expression
  THEN BEGIN
    true actions
    END
  ELSE BEGIN
    false actions
    END
```

The key words, BEGIN and END, serve the same function as the braces do in C. The boolean expression need not be enclosed in parentheses. The same need to group actions together exists for the Pascal constructs WHILE, REPEAT, FOR, PROCEDURE, and FUNCTION.

A common feature of most languages is the use of keywords and/or some sort of brackets or braces to delimit the various elements that form the pieces of these control structures.

### PISTOL *Structures*

PISTOL is no exception. It groups the elements or actions by using pairs (and occasionally triplets) of keywords. The following elements are defined as shown:

1. Procedure or function:
   'name : body of definition ;
2. Two-way branch (the "else" clause optional):
   condition IF true actions ELSE false actions THEN
3. FOR loop:
   upper-lim initial-val DO body LOOP
4. REPEAT loop:
   BEGIN actions condition END
5. WHILE loop:
   BEGIN condition IF actions REPEAT

The "case" construction (actually closer to the COND of LISP) is for the benefit of humans:

```
variable OFCASE
        condition C: actions; C
        condition C: actions; C
        .
        .
        .
        ENDCASE
```

### PISTOL's *Syntax Checking*

During compilation, the PISTOL compiler is particularly careful to verify that the user is following the few syntactical rules that do exist. To test for proper nesting, the compilation process uses a check stack. When a word that starts a structure is encountered, a character of the appropriate type is placed on this stack, and when a word that completes a structure is encountered, a character is removed from the stack and checked to make sure it is of the right type.

Consider the following fragment of

PISTOL code:

```
... : ... DO ... BEGIN ... END ...
        LOOP ... ;
```

In this example, the : leaves a ":" on the check stack; the DO leaves a "D" on the stack; and the BEGIN leaves a "B" on the stack. When the END is encountered, the check stack is popped (returning the "B" from BEGIN) to see whether a BEGIN was used last. Later the LOOP is encountered, and the stack is popped and checked for "D." Lastly, the ; is encountered, so the popped character is checked to be a ":."

Not only does the check stack ensure proper nesting has been maintained (an incorrect nesting is likely to produce code that would catastrophically crash), but it also enables the compiler-interpreter to ascertain when an input line completes the outermost structure; it may then proceed to interpretation (execution) of the code.

As an aid to the user, PISTOL prompts with a display of the current contents of the check stack. The user sees exactly into which structures he or she has entered and still must complete.

### Brackets Proposal

I propose that a new structure be available to Forth-like languages: braces (curly brackets). They would be optional but could be inserted for improved clarity. Let's examine an example taken from one of the basic definitions in PISTOL (from the file PBASE2):

```
'INDENT : DUP TERMINAL-
          WIDTH W@ LT IF
          COLUMN W@ -
          SPACES
          ELSE IFCR DROP
          THEN ;
```

With the addition of braces; we would have:

```
'INDENT : DUP { TERMINAL-
          WIDTH W@ LT } IF
          { COLUMN W@ -
          SPACES } ELSE
          { IFCR DROP } THEN
          ;
```

Here the readability is improved in that what IF is testing is delineated as well as the two alternative conditional courses of action. Forth-like languages are sometimes criticized because everything runs together; could these braces form the solution?

Another benefit of braces is the deferral of both execution and the recycling of string space. For example:

```
X> "HELLO THERE" MSG
HELLO THERE
```

properly prints the greeting, whereas:

```
X> "HELLO THERE"
1X> MSG
MSG
```

obviously does not! In the second example, the "HELLO THERE" pushes a pointer to itself on the parameter stack. (We see from the second prompt that there is now one item on the stack.) However, the pointer points to an area where the space has been recycled, so the (erroneous) string "MSG" is printed instead. With braces this recycling action can be delayed, as follows:

```
X> {
X{> "HELLO THERE"
X{> MSG
X{> }
HELLO THERE
```

The prompt indicates that we have entered the "brace structure." Thus we need not worry about ending lines after we have typed in (possibly quite long) strings needed for execution on the following lines; we simply must remember to encircle the strings and the words that use them with braces.

I have included a listing of a file BRACKET that adds the needed definitions to this structure (Listing One, page 104). Note how little work is needed to extend Forth-like languages!

I also have included the transcript of a session at the terminal that shows what happens when BRACKET is loaded, as well as a number of facilities that are part of PISTOL (Listing Two, page 106).

### Conclusion

Forth-like languages such as PISTOL demonstrate that delimiters such as parentheses or braces are not required for a general-purpose computer language. (Such languages, however, may use them for other things. Forth implementations, for example, often use parentheses and square brackets for special purposes.) With the recent emphasis on readability by people even at the expense of greater effort in writing, perhaps braces should be available for optional use by the writer. The language compiler could ignore these braces except perhaps to check the syntactical requirements that structures properly "nest," as with PISTOL here. Whether or not such a feature can be added to a particular Forth system as easily as it has been added here to PISTOL probably depends on details of the particular implementation.

*PISTOL is available on a CP/M diskette at nominal cost from the C User's Group, Box 287, Yates Center, KS 66783, and as SIG/M Volume 126 from the Amateur Computer Group of New Jersey (ACGNJ), Box 319, South Bound Brook, NJ 08880.*

**BBJ**

**Reader Ballot**
Vote for your favorite feature/article.
Circle Reader Service **No. 195.**

# Languages and Parentheses (Text begins on page 102)

## Listing One

```
% OCTOBER 2, 1983 ****BRACKET DEFS FOR PISTOL*****
% THE TOKENS, { and } ARE NOPS IN USE BUT ENABLE
% THE PROGRAMMER TO SUPPLY EMPHASIS TO GROUPINGS

HEX

% IMMEDIATE PATCHES DEFINITION TO EXECUTE DURING COMPILATION:
'IMMEDIATE : 'NOP ADDRESS CURRENT W@ W@ W- W! ;

'SYNTERR : "SYNTAX ERROR" MERR ;

'{ : SYNTAXBASE C@ 1+ DUP CHKLMT LT
     IF SYNTAXBASE C!                    % UPDATE COUNT
        7B SYNTAXBASE DUP C@ + C!        % "PUSH" '{'
     ELSE SYNTERR
     THEN
   ; IMMEDIATE

'} : SYNTAXBASE DUP C@ + C@ 7B EQ
     IF SYNTAXBASE C@ 1- SYNTAXBASE C!
     ELSE SYNTERR
     THEN
   ; IMMEDIATE

;F

        Listing of BRACKET, which adds the definitions of "{"
and "}" to PISTOL.
```

End Listing One

## Listing Two

The enclosed terminal session listing is provided to demonstrate a number of important features and aspects of PISTOL.

Here is a running commentary on that listing. The only hint that this was run on a CP/M environment appears on the first line with the system prompt A>. (All subsequent dialogue is standard for all PISTOL environments.) Restoring the "core image," CORE2, is a fast way to recreate PISTOL without recompiling the extensive source code that is supplied originally to bring PISTOL up. Turning the ECHO ON enables us to watch the lines of the file

*(Continued on next page)*

## Listing Two

BRACKET being input for compilation.

BRACKET is being loaded; the following lines should look very much like the supplied listing of BRACKET, but here PISTOL is supplying the prompts. The word % is used to mark the rest of the line as a comment. Notice that after a couple of lines the prompt changes from X> to H> to show that the number base has been switched from decimal to hexadecimal (by the word HEX).

The word IMMEDIATE is defined to poke the address of NOP into a location that has to be calculated with respect to the start of the code of the CURRENT definition. The operator W@ does a word fetch; the W- backs up an address by one word size; W! pokes the word (here, the address of NOP).

After SYNTERR is defined we see { being defined. Since the definition takes several lines, we see that the prompt contains a : , indicating that the : ... ; structure is incomplete; also, within the same definition, an IF...ELSE...THEN structure is used and the "F" and "E" tokens appear in the prompt. The number that starts the prompt indicates that the parameter stack is not empty; it is used in the compiling process to calculate the address offsets needed to generate the code associated with most of the structures.

The word ;F is the logical end-of-file; PISTOL confirms that the loading of the file is complete.

I have typed TOP10 to get a list of the 10 most recent definitions. Since I am not interested in seeing the next 10 most recent definitions, I throw away (DROP) the backward link left on the stack. DECIMAL returns me to base 10.

It is possible to disassemble a definition with the built-in disassembler DIS; IMMEDIATE is analyzed to look more or less like its original definition.

Finally, I do a few tests of the definitions of { and of }. It does appear that the prompt is properly indicating the number of brackets, and when I type too many }'s, an error message is displayed. Also checked here is the way strings are recycled and that the { can be used to defer execution until the matching } is typed.

Lastly, the TRACE facility is demonstrated for the word MSG. BYE is the proper way to exit PISTOL.

```
A>PISTOL
*** PISTOL 2.0 ***
X>'CORE2 RESTORE
X> ECHO ON
X> 'BRACKET LOAD
X> % OCTOBER 2, 1983 ****BRACKET DEFS FOR PISTOL*****
X> % THE TOKENS, { and } ARE NOPS IN USE BUT ENABLE
X> % THE PROGRAMMER TO SUPPLY EMPHASIS TO GROUPINGS
X>
X> HEX
H>
H> % IMMEDIATE PATCHES DEFINITION TO EXECUTE DURING COMPILATION:
H> 'IMMEDIATE : 'NOP ADDRESS CURRENT W@ W@ W- W! ;
H>
H> 'SYNTERR : "SYNTAX ERROR" MERR ;
H>
H> '{ : SYNTAXBASE C@ 1+ DUP CHKLMT LT
1H:>       IF SYNTAXBASE C!                  % UPDATE COUNT
2H:F>          7B SYNTAXBASE DUP C@ + C!     % "PUSH" '{'
2H:F>       ELSE SYNTERR
2H:E>       THEN
1H:>     ; IMMEDIATE
H>
H> '} : SYNTAXBASE DUP C@ + C@ 7B EQ
1H:>       IF SYNTAXBASE C@ 1- SYNTAXBASE C!
2H:F>       ELSE SYNTERR
2H:E>       THEN
1H:>     ; IMMEDIATE
H>
H> ;F
BRACKET LOADED
H>
H> TOP10
```

*(Continued on page 108)*

## Listing Two

```
687C    }
684A    {
683A    SYNTERR
6820    IMMEDIATE
67EE    HELP
66DA    TYIL
668A    UC
667C    Q'
666E    A'
65EC    FINISH
1H> DROP
H> DECIMAL
X> 'IMMEDIATE DIS
'IMMEDIATE [:]  "NOP"    ADDRESS CURRENT W@      W@      W-
W!      ;
X>
X> {

X{> { { {

X{{{{> } }

X{{> }

X{> }
X> }
SYNTAX ERROR
*** PISTOL 2.0 ***
X> "HELLO THERE" MSG
HELLO THERE
X> "HELLO THERE"
1X> MSG
MSG
X> { "HELLO THERE"

X{> MSG

X{> }
HELLO THERE
X> 'HELLO 'MSG TRACE
'MSG BEING TRACED:
(1)   32086
(2)   32086   32086
(2)   32086   5
(1)   32086
(2)   32086   32086
(2)   32086   32087
(2)   32087   32086
(2)   32087   5
(0)
TRACE COMPLETED
X> BYE

PISTOL NORMAL EXIT
```

```
DUP
C@
LINE-SPACE?
DUP
1+
SWAP
C@
TYPE   HELLO
(;)
```

End Listings

# RED

edward k. ream

## FULL SCREEN EDITOR
with
## FULL SOURCE CODE in C
for
## CP/M 68K or CP/M 80

- RED is a powerful yet simple full screen text editor for both programmers and writers.
- RED features 17 commands including block move, block copy, search and substitute.
- RED comes with *full* source code in standard C. RED works as is with the BDS C, Aztec CII and Digital Research Compilers.
- RED supports *all* features of your terminal. You tailor RED to your terminal with an easy-to-use configuration program.
- RED handles files as large as your disk.
- RED is guaranteed. If for any reason you are not satisfied with RED, your money will be refunded promptly.

## Price: $95.

**Call today for more valuable information:**
**(608) 231-2952**

**To order, send a check or money order to:**

**Edward K. Ream**
**1850 Summit Avenue**
**Madison, Wisconsin 53705**

Your order will be mailed to you within one week. Sorry, I can not handle credit cards. Please do not send purchase orders unless a check is included. RED is distributed only on 8 inch CP/M format disks, but other disk formats are available through third parties.

*Dealer inquiries invited.*

---

**GGM – FORTH™ has HELP***
**for Z80[1] using CP/M[2]**

**GGM–FORTH,** a complete software system for real-time measurement and control, runs on any Z80 computer under CP/M using an extended fig-FORTH vocabulary.

**GGM–FORTH** features:

- Open multiple CP/M files, in any combination of direct-access and sequential-access, fully compatible with all CP/M utilities
- Char. in/out uses CP/M console, lister, file, or port
- On-line HELP* provides instant access to definitions in the run-time GGM–FORTH dictionary
- HELP* file is easily extended to include user definitions using HELP* utility
- HELP* is available during full-screen editing

| | |
|---|---|
| Complete system and manuals | $150. |
| Manuals only: | $ 20. |
| Introductory System: | $ 35. |

GGM SYSTEMS, INC. (617) 662-0550
135 Summer Ave., Reading, MA 01867

[1] Z80 is a trademark of Zilog, Inc.
[2] CP/M is a trademark of Digital Research, Inc.

Circle **no. 29** on reader service card.

---

**At Last! bds C . . .**

# Ver.1.5

**Including a new dynamic debugger**
**Still the choice of professionals**

- Compiler option to generate special symbol table for new dynamic debugger by David Kirkland. (With the debugger, the distribution package now requires two disks.)
- Takes full advantage of CP/M® 2.x, including random-record read, seek relative to file end, user number prefixes, and better error reporting.

**V 1.5 . . . . . $120.00**
**V 1.46 . . . . $115.00**
(needs only 1.4 CP/M)

- Clink option to suppress warm-boot
- New library file search capabilities
- New, fully-indexed 180 page manual
- ® CP/M is a trademark of Digital Research, Inc.

Other C compilers and C related products available . . . Call!

TERMS: CHECK, MONEY ORDER, C.O.D., CHARGE CARD
HOURS: 9 am—5 pm Monday —Friday
**(316) 431-0018**

IT'S HERE!
**MONEY MATH**
- Uses BCD internal representation.
- You choose from two types of rounding.
- Configurable exception handling
- Distributed with 12 digits precision. Easily configured for more or less
- Excess 64 exponents

**SOURCE INCLUDED** **$50.00**

DMS
Dedicated Micro Systems, Inc.
P.O. Box 481, Chanute, Kansas 66720

**include $2.50 for postage and handling**

Circle **no. 25** on reader service card.

# Balancing Act:

## The Ultimate Checkbook Balancing Program

Balancing a checkbook is a little like asking which is the best programming language: every time you do it, you're liable to get a different answer. But now, after long years of expensive research, that problem is behind us. This article won't just describe what may be the world's easiest-to-use checkbook balancing program—the revolutionary program itself is included here free. (See the listing below.)

Everyone seems to have trouble with checkbooks and banks. Do you find yourself struggling with Stith's first law of banking: No Deposit, No Return? Do you wonder how banks can make you stand in line and then demand a service charge? How often are you told "The check is in the mail"? Have you ever received a dirty look from a bank guard when you've joined a long line and asked, "Hey, what's the holdup?"? If so, a stupid program isn't going to solve anything. But if you just need an infinitely simpler way to balance your checkbook, read on.

by John E. Stith

*John E. Stith, P.O. Box 6677, Colorado Springs, CO 80934.*

This program has all the traditional boring features. It runs rapidly, uses minimal storage space, has adequate comments, and is easy to convert to other BASICs. (It's written in Microsoft BASIC.) But I didn't stop there. No.

Not only does it work with paper *and* rubber checks, you never again have to record a checking transaction. In fact, you don't even need an active account. How many programs require rerunning every single month? Not this one. Run it once and you're done—it does it right the very first time.

It also runs perfectly with personalized checkbooks, as long as your name is printed in lightweight letters. Cowhide, pigskin, hamster hide—no problem. It works with both rectangular and square checkbooks. An ambitious clever programmer could probably even modify it to handle *round* checkbooks. Finally, the new, improved version 17.0 adapts automatically to whatever units you prefer. Enter your dimensions in inches, microns, light years, and the program will deliver your answer in those very same units.

So, say goodbye to early withdrawal symptoms and automated tellers that don't smile back at you. Here's your chance to let modern technology solve more problems than it creates.

**DDJ**



## Balancing Act Listing

```
100 REM BALANCING ACT -- THE ULTIMATE CHECKBOOK BALANCING PROGRAM
110 REM AUTHOR -- JOHN E. STITH -- c COPYRIGHT 1983
200 REM L = CHECKBOOK LENGTH
210 REM W = CHECKBOOK WIDTH
300 PRINT "BALANCING ACT -- THE ULTIMATE CHECKBOOK BALANCING PROGRAM"
310 PRINT
400 PRINT "WHAT IS THE LENGTH OF YOUR CHECKBOOK (THE LONG DIMENSION)";
410 INPUT L
500 PRINT "WHAT IS THE WIDTH OF YOUR CHECKBOOK (THE NARROW DIMENSION)";
510 INPUT W
550 PRINT
600 PRINT "NOW PLACE YOUR CHECKBOOK SO ITS LONG DIMENSION FACES YOU."
610 PRINT "MARK THE SPOT"; L/2 ; "UNITS FROM THE LEFT SIDE ";
620 PRINT "AND"; W/2 ; "UNITS FROM THE TOP."
630 PRINT "THIS SPOT IS AS CLOSE AS YOU ARE GOING TO GET TO ";
640 PRINT "THE CENTER OF GRAVITY."
650 PRINT "PLACE YOUR PENCIL POINT UNDER IT, ";
660 PRINT "AND YOUR CHECKBOOK SHOULD BALANCE."
670 PRINT "DON'T WORRY IF IT DOESN'T BALANCE PERFECTLY.  ";
680 PRINT "EVERYONE HAS THAT PROBLEM."
999 END
```

**End Listing**

## New Release 4.1

**Microstat®**

We've continually improved Microstat since it was introduced in 1978, and the latest release includes many new features you've wanted.

- Interactive and Batch Processing
- Expanded Data Management Subsystem with New Data Transforms
- Reading data files created by other programs
- 3 types of Analysis of Variance
- Time Series
- Crosstabs and Chi-Square
- Factorials, Permutations, and Combinations
- Hypothesis Tests
- Data sets that can exceed memory
- Multiple Regression (including Stepwise)
- Scatterplots (including best fit regression)
- Correlation Analysis
- 12 Nonparametric tests
- 8 Probability Distributions
- Descriptive Statistics
- Easy Installation

Microstat's algorithms have been designed to prevent numeric overflow errors and yield unsurpassed accuracy. Microstat's price is $375.00 including the user's manual and is available for the Z80, 8086, 8088 CPU's and CP/M80, CP/M86, MS-DOS, and PC-DOS. To order, call or write.

ECOSOFT INC.

**6413 N. College Ave. • Indianapolis, IN 46220**
**(317) 255-6476**

VISA MasterCard

Trademarks: Microstat (Ecosoft), CP/M (Digital Research), MS-DOS (Microsoft), PC-DOS (IBM), Z80 (Zilog), 8086, 8088 (Intel).

Circle **no. 27** on reader service card.

# LISP
## FOR THE IBM PERSONAL COMPUTER.

### THE PREMIER LANGUAGE OF ARTIFICIAL INTELLIGENCE FOR YOUR IBM PC.

- **DATA TYPES**
  - Lists and Symbols
  - Unlimited Precision Integers
  - Floating Point Numbers
  - Character Strings
  - Multidimensional Arrays
  - Files
  - Machine Language Code
- **MEMORY MANAGEMENT**
  - Full Memory Space Supported
  - Dynamic Allocation
  - Compacting Garbage Collector
- **FUNCTION TYPES**
  - EXPR/FEXPR/MACRO
  - Machine Language Primitives
  - Over 190 Primitive Functions
- **IO SUPPORT**
  - Multiple Display Windows
  - Cursor Control
  - All Function Keys Supported
  - Read and Splice Macros
  - Disk Files
- **POWERFUL ERROR RECOVERY**
- **8087 SUPPORT**
- **COLOR GRAPHICS**
- **LISP LIBRARY**
  - Structured Programming Macros
  - Editor and Formatter
  - Package Support
  - Debugging Functions
  - .OBJ File Loader
- **RUNS UNDER PC-DOS 1.1 or 2.0**

**IQLISP**

| | |
|---|---|
| 5¼" Diskette and Manual | $175.00 |
| Manual Only | $ 30.00 |

*Integral Quality*

P.O. Box 31970
Seattle, Washington 98103-0070
(206) 527-2918

Washington State residents add sales tax.
VISA and MASTERCARD accepted.
Shipping included for prepaid orders.

Circle **no. 33** on reader service card.

*Introducing*

# WALTZ LISP™

The one and only adult Lisp system for CP/M users.

Waltz Lisp is a very powerful and complete implementation of the Lisp programming language. It includes features previously available only in large Lisp systems. In fact, Waltz is substantially compatible with Franz (the Lisp running under Unix), and is similar to MacLisp. Do not be deceived by the low introductory price.

Waltz Lisp is a perfect language for Artificial Intelligence programming. It is also suitable for general applications. In fact, due to the ease of handling of textual data and random file access functions, it is often easier to write a utility program in Waltz Lisp than in any other programming language. Several general purpose utilities (including **grep** and **diff**) written entirely in Waltz Lisp are included with the interpreter.

Much faster than other microcomputer Lisps. • Long integers (up to 611 digits). Selectable radix. • True dynamic character strings. Full string operations including fast matching/extraction. • Random file access. • Binary files. • Standard CP/M devices. • Access to disk directories. • Functions of type lambda (expr), nlambda (fexpr), lexpr, macro. • Splicing and non-splicing character macros. • User control over all aspects of the interpreter. • Built-in prettyprinting and formatting facilities. • Complete set of error handling and debugging functions including user programmable processing of undefined function references. • Optional automatic loading of initialization file. • Powerful CP/M command line parsing. • Fast sorting/merging using user defined comparison predicates. • Full suite of mapping functions, iterators, etc. • Over 250 functions in total. • Extensive manual with hundreds of illustrative examples.

Waltz Lisp requires C/PM 2.0, Z80 and 48K RAM (more recommended). SS/SD 8" and most common 5" disk formats.

**Introductory Price....$94.50**

Manual only .....$20.00
(refundable with order)

additional charges
$10.00 conversion fee for 5" Diskettes
$3.00 C.O.D. charge

**Call toll free 1-800-LIP-4000** Ask for Dept. #3
In Oregon and outside U.S.A. call 1-503-684-3000

Unix® Bell Laboratories. CP/M® Digital Research Corp.

PRO CODE™ INTERNATIONAL

P. O. Box 7301
Charlottesville, VA 22906

Circle **no. 50** on reader service card.

## C Programmer's Library

**by Jack Purdum, Timothy Leslie, Alan Stegemoller**
**Published by Que Corporation**
**$19.95, 366 pages**
**Reviewed by John R. Johnson**

The definitive source on the C programming language is, and will likely remain, the book *The C Programming Language* by Brian Kernighan and Dennis Ritchie. It is elegant, complete, and spare. Needless to say there has been a rush to publish expansions, explanations, and elucidations.

One of the best of these was the *C Programmers Guide* by Jack Purdum. This second book also issues from the senior staff at Ecosoft Incorporated, and seems to continue where the first left off.

The declared audience for the book is a programmer who is familiar with the syntax of the C language and wishes to learn how to develop the full power of C. Rather as a bonus, they throw in the sample source code they use to demonstrate proper use of declarations, structures, and functions for library construction. This is code for an ISAM data base system with a significant application example, a book catalog and retrieval system.

It is apparent that several authors were involved. The initial coverage of complex declarations was thorough and well done. I thought it a good idea to tie the declaration followthrough to an actual compiler algorithm. A good understanding of the underlying mechanism makes variations in syntax more understandable.

The coverage of structures, on the other hand, was much more subtle. Rather than just saying "here is a structure" the author let you become aware of what using structures correctly does for your programs. This directed self-discovery is also a powerful tool for communication of understanding. The chapter on sorting is good, not for the sorting information, but for the separation of functions. The reader is shown how to improve average programs by taking advantage of the features of the C language.

Although all of the authors are associated with the same company, which does produce a C compiler system, you would not know that from the treatment. They have an appendix with samples of the library functions that have to be added to compile the sample programs with most of the popular C compilers available for micros. As an added bonus most of the sample programs were compiled and tested with the UNIX PCC compiler.

Overall, I found the book a useful adjunct to my Kernighan and Ritchie. I think that most serious users of C would find it worth having the samples in a program library and the book on the shelf.

## Building Controls Into Structured Systems

**by Alan E. Brill**
**Published by Yourdon Press**
**Reviewed by Dr. Joseph B. Rothstein**

As microcomputer programming has moved from the basements of hackers to the boardrooms of the Fortune 500, the demand for accountability and control for microcomputer programs has grown almost as fast as the microcomputer industry itself.

The thought that programs run on micros would be important enough to warrant auditing would have drawn laughter just a few years ago. But as successive generations of ever-more-powerful microcomputers have become an essential part of corporate life, comptrollers and financial managers have come to insist on the same sort of tight controls and auditing procedures they are used to on their mainframe and minicomputer systems.

The systems analysts, auditors, and financial management specialists at Yourdon, Inc. have established themselves among the leading consultants to big business. They embraced the concepts of structured programming early on, and have built a reputation on their seminars, periodicals, and book titles.

A newly published text from Yourdon Press, *Building Controls Into Structured Systems* by Alan E. Brill condenses some of the most important concepts of EDP auditing, applications systems controls, and essential procedures into fewer than 150 pages. While not aimed specifically at micro users or programmers (specific hardware is not even mentioned), the concepts involved are as appropriate to a "Little Wonder-80" as they are to an IBM 4300.

This is not an accounting book; nor is it a book about programming — indeed, there is not a single line of program code in the entire volume. Rather, Brill describes a methodology for building well-controlled systems featuring internal accounting controls and external application controls, all designed to provide reasonable assurance that the system is able to detect and handle any errors or anomalies that may crop up.

Numerous examples, including fictional "dialogues" between programmers and users, auditors, or managers, illustrate the importance of such controls. As more and more businesses automate their payrolls, customer lists, etc., the risks of uncontrolled systems are all too often illustrated by fiscal tragedy. Such horror stories seldom make headlines; most businesses would rather take their lumps and swallow their losses than have the world learn how lax they've been. But most programmers in large organizations can describe in gory detail how some other guy (never themselves!) resigned in disgrace after his or her system fell

# C Programmers: Program three times faster with *Instant-C*™

***Instant-C***™ is an optimizing **interpreter for C** that makes programming three or more times faster. It eliminates the time wasted by compilers. Many repetitive tasks are automated to make programming less tedious.

- Two seconds elasped time from completion of editing to execution.
- **Symbolic debugging;** single step by statement.
- Compiled execution speed; 40 times faster than interpreted Basic.
- Full-screen editor integrated with compiler; compile errors set cursor to trouble spot.
- Directly generates .EXE or .CMD files.
- Follows K & R—works with existing programs. Comprehensive standard C library with source.
- Integrated package; nothing else needed.
- Works under PC-DOS*, MS-DOS*, CP/M-86*.

More productivity, less frustration, better programs. ***Instant-C***™ is $500. Call or write for more info.

**Rational Systems, Inc.**

(617) 653-6194
P.O. Box 480
Natick, Mass. 01760

Trademarks: MS-DOS (Microsoft Corp.), PC-DOS (IBM), CP/M-86 (Digital Research, Inc.), Instant-C (Rational Systems, Inc.)

Circle **no. 59** on reader service card.

# FOR $29.95, *MICRO-WYL* WILL WHEN OTHERS WON'T!

You'll know why INFOWORLD rated *Micro-WYL's* performance "Excellent" when you want to insert a file into a file; when you want to print directly from the screen; when you want to see what your text will look like before printing; set tabs where you want them; write programs the easy way; and do many other things that no other word processor can do. *Micro-WYL* is a line editor; use it in conjunction with EMACS derivative word processors and you have the best of all worlds! For $29.95, plus $2.00 postage and handling, have it your way! Sold with the usual Overbeek 30 day money-back guarantee.

LET *Micro-WYL* DO IT ALL TODAY!
Enclosed find check for $31.95 or

charge my Mastercard# _____ Expires: _____

Visa# _____ Expires: _____

**Check format desired:**

| | | |
|---|---|---|
| __8" SSSD | __Osborne Single Density | __KayPro II |
| __Superbrain | __Osborne Double Density | __NEC 5" |
| __Northstar Advantage | __Morrow Micro Decision | __Televideo802 |
| __Northstar Horizon | __Xerox 820 Single Density | __ALTOS 5 |
| __Apple/Softcard | __Xerox 820 Double Density | __Epson |

I'm not ready to order now, but send me information about all the affordable programs from Overbeek Enterprises.

Name _____
Address _____
City _____ State _____ Zip _____

OVERBEEK ENTERPRISES, P.O. Box 726D, Elgin, IL 60120
312-697-8420

*Micro-WYL—another affordable program from Overbeek Enterprises.*

Circle **no. 46** on reader service card.

# Z80* SINGLE BOARD COMPUTER!

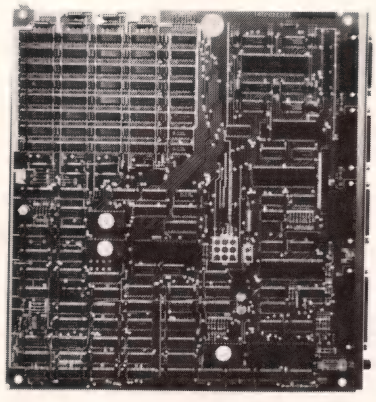
## 64K RAM — 80 x 24 VIDEO DISPLAY — FLOPPY DISK CONTROLLER

## RUNS CP/M* 2.2!

BRAND NEW PC BOARDS WITH SCHEMATICS!

BOARD MEASURES 11½" x 12½"

ALL ORDERS WILL BE PROCESSED ON A *STRICT*, FIRST COME, FIRST SERVED BASIS! ORDER EARLY!



**$29.95** (BLANK BOARD WITH DATA AND ROM'S.)

**NEW PRICE**

GROUP SPECIAL: BUY 6 FOR $165!

USES EASY TO GET PARTS!

## *UNBELIEVABLE LOW PRICE!!! GIANT COMPUTER MANUFACTURER'S SURPLUS!*

Recently Xerox Corp. changed designs on their popular 820* computer. These prime, new, 820-1 PC boards were declared as surplus and sold. Their loss is your gain! These boards are 4 layers for lower noise, are solder masked, and have a silk screened component legend. They are absolutely some of the best quality PC boards we have seen, and all have passed final vendor QC. Please note, however, these surplus boards were sold by Xerox to us on an *AS IS* basis and they will not warranty nor support this part.

We provide complete schematics, ROM'S, and parts lists. If you are an *EXPERIENCED* computer hacker, this board is for you! Remember, these are prime, unused PC boards! But since we have no control over the quality of parts used to populate the blank board, we must sell these boards as is, without warranty. You will have to do any debugging, if necessary, yourself!

*CP/M TM OF DIGITAL RESEARCH INC. (CALIF.) 820 TM OF XEROX CORP. Z80 TM OF ZILOG

WE ALSO CARRY LS, Z-80, EPROM'S, ETC. SEND FOR FREE CATALOG!

ADD $2 PER PC BOARD FOR SHIPPING. (USA and Canada)

**B. G. MICRO**

P. O. Box 280298 Dallas, Texas 75228
(214) 271-5546

TERMS: Orders over $50 add 85¢ insurance. No COD. Tex. Res. Add 6% Sales Tax. Subject to prior sale. Foreign orders: US funds only. We cannot ship to Mexico. Foreign countries other than Canada add $6 per board shipping.

Circle **no. 9** on reader service card.

apart. It's just this sort of tragedy that the author counsels programmers how to avoid.

As the title implies, structured techniques are emphasized throughout. Just as these techniques can be applied to the task of designing and implementing program code, so can they be applied to auditing and application controls. In the first few chapters Brill introduces the field of EDP auditing and the concepts of phase-related control — the process of identifying, specifying, and documenting the internal controls appropriate to each stage of the system's development life cycle. Successive chapters cover modeling controls during the analysis, design, and implementation phases, followed by chapters on documentation reviews and maintenance reviews. The book closes with a discussion of several typical plans for action that are almost sure to fail, then contrasts those with an approach that is more likely to succeed.

A breezy conversational style is used throughout, enlivening a subject that could easily have been treated in an academic and jargonistic fashion. Drawings, charts, and summaries are used extensively to further clarify important concepts.

Clearly, though, this is not a book for everyone. The hobbyist, game designer, or personal-software user may never need or want such stringent controls. But once money enters the picture, especially big money — and particularly *corporate* big money — the controls and audit procedures Brill describes are likely to become *de rigueur*. The serious professional programmer had best acquaint him or herself with them.

As the author states in his Preface, "You *will* eventually have to put controls into your application systems. You can choose to wait until the auditors and your boss force you to do so, but wouldn't it be easier and more impressive if you took the initiative to *assure* that effective internal controls are a built-in part of every application system that you build?" It's just this sort of no-nonsense, practical approach that makes *Building Controls Into Structured Systems* a worthwhile introduction to a subject that is moving to the forefront among the concerns of business executives and EDP planners, and thus will influence the careers and futures of programmers as well.



## MENTOR — The Magazine on Disk

**Mentor Computer Series, 533 Sutter Street, Suite 914, San Francisco, CA 94102**
**$18.00 per issue**
**Reviewed by Dian Crayne**

The idea of putting magazines on disk is one of those notions that have been kicking around, under one guise or another, ever since computers came out of the office and into the home.

Most publishing companies have shied away from the diskette format because of the problems involved. Not only would they have to go through all of the monumental work involved in putting out a magazine, they would also have to make sure that it got transferred (without errors) to a flexible, rather fragile diskette and sent to a subscription list without being folded, spindled, mutilated, or erased. The idea alone is enough to make most publishers turn pale.

One company that has faced up to the challenge is Mentor Computer Services, which published the fourth issue of *Mentor,* "the magazine on disk," in January of this year. Although some other publishers — notably Ziff-Davis with its *PC Disk* — have flirted with the concept of sending out software on magnetic media, *Mentor* appears to be the only company that has really gotten off of the ground so far.

"Disk is more appropriate media than paper for computers," says editor Ted Lester, whose staff does both editing and publishing for this venture. He went on to explain that he is particularly proud of the fact that *Mentor* has published some excellent add-on utilities for existing programs, which gives subscribers software they can actually use instead of curiosities to stuff away on a shelf.

The first issue of *Mentor,* for instance, carried a WordStar customizing program; follow-up articles appeared in the next two issues. Issue No. 4 also carried LEADS, "The Buyer's Worksheet for Lotus 1-2-3," a worksheet program that helps salesmen to track customers and sources. *Mentor* has also published mailing list programs and a MailMerge enhancement package.

Other articles and programs that appeared in issue No. 4 were "Critical Path Method Project Schedule," by Philip Jacka; "Tally," a data base counting program by Richard Malm; a discussion on "Transfers Between dBase II and 1-2-3" by LeBlond and Cobb; and "Clock," an assembly language tutorial program that shows users how to display the current time on their screen.

There was also an article on PCrayon, a discussion of job streaming for Pascal compilers, and some regular features including a column by Rick Albert, letters to the editor, editorials, and a shoppers' reference list to products mentioned in the magazine. And yes, there is a cumulative index.

*Mentor* comes in a sturdy black plastic diskette box, the kind that opens up to make a vertical file for the diskettes. New subscribers get a startup diskette and one or more program diskettes, which hold the contents of the magazine. Since DOS is not distributed with the magazine (diskzine?), users have to create their own system diskette and then move the required files to it from the startup diskette. Once you've gotten the starter set, subsequent issues consist only of the magazine diskettes themselves.

*Mentor* looks a lot like a hard-copy magazine once you begin to display it on the screen. The first page displays the magazine logo, and the next few pages list the editorial staff and then the contents. Pages are turned by pressing the return (enter) key, and you can either page through the entire magazine or read the table of contents and select an article by its letter. There are even some advertisements, but unlike a lot of regular computer magazines on the stands, *Mentor*'s articles aren't submerged in an ocean of commercial ballyhoo.

Once you've selected an article to read, you can page through it, abandon it and select another one, or — if the article is written in conjunction with a piece of software — actually run a program and get an immediate demonstration of the way it works. (The editorial column in issue No. 4 lists the code for a short animation routine, executes the routine, and then returns to a discussion of what happened.) This is the sort of immediate reinforcement that educators have been pursuing for years.

Editor Lester regards this ability to run programs within the boundaries of *Mentor* as one of the most exciting facets of the magazine. He has plans to take the magazine's educational aspects even further by establishing a subscriber's remote bulletin board system, where programming techniques and new advances in electronics can be discussed in open forum.

Another advantage *Mentor* has over traditional computer magazines is the storage space it saves: *Mentor*'s hard 6½-inch square boxes take a lot less room on the shelf. Given a box roughly half the thickness and half the height of some of the larger computer magazines, you can store a lot more magazines in the same space. Since they take less space, subscribers are more apt to keep them, which means less agony from realizing that a particular article went out in the trash two months ago because the garage filled up.

Although *Mentor* is informative and many of the programs included with it are undeniably useful, the magazine itself is visually unexciting. Advertisements are simple text blocks, and no large headers are used for the articles. Inclusion of a few simple graphics — even graphics built out of text characters to maintain compatibility with IBM Monochrome monitors — would go a long way towards brightening the magazine's image.

This lack of a good visual image is one that strikes right at the heart of advertising. Most magazines subsidize their publishing costs through sales of advertising, and major advertisers have traditionally been geared toward full-page color ads. It remains to be seen whether they will tackle display screen formats, although the concept does bring up some interesting opportunities for the future, when advertisements may consist of full-color animated sequences complete with sound.

One enhancement that *Mentor* definitely needs is some sort of hard-copy table of contents. At present the only way you can see what is in each issue is to mount it in your drive and take a look. Perhaps the *Mentor* people could include a large square label for subscribers to attach to each issue's box or simply print the table of contents on the diskette label.

Minimum system requirements for using *Mentor* are a 64K IBM PC system with a monochrome or color display and at least two single-sided drives. However, a special PCjr edition that requires only one drive is in preparation and may be available by summer of 1984. The *Mentor* staff expects to be publishing a good deal of PCjr software in the coming issues, including color graphics, animation, and entertainment programs.

The editors of *Mentor* have latched onto an idea that has almost limitless opportunities for software distribution, education, and entertainment. It will be interesting to watch this diskzine over the next couple of years to see how the editors and publishers handle their myriad challenges.

■■J

by Ray Duncan

## iRMX-86 for the IBM PC

Intel's iRMX-86 operating system for the 8086 series of microprocessors is a far cry from MS-DOS or CP/M-86. It is a real-time, multi-tasking, multi-user operating system with an excellent assembler and linker and an extensive arsenal of classy, optimizing compilers. The high-level languages available include PL/M-86, 87PASCAL, and 87FORTRAN—all support the "large memory model," reentrant code, direct control of the hardware . . . and, as you might have guessed from the names, all of the languages fully support the 8087 numeric coprocessor.

The Intel assembler, ASM86, is powerful, fast, and very polished. The macro facilities are solid and extensive; the Code-Macro facility is also present that allows you to add new opcodes to the assembler. A particularly nice feature is that you can link your programs with either a true 8087 function library or an 8087 emulator library without changing a word of the source code.

iRMX-86 has layers upon layers of interfaces and features. Your programs can talk to the operating system through calls to the Nucleus Basic I/O System, Extended I/O System, or UDI—each level has its own set of executive services and calling conventions. An additional layer, the Human Interface, contains the command utilities for formatting and copying disks, reading directories, managing files, and loading application software.

With all of this power, of course, comes considerable complexity. There are about a thousand pages of documentation or so just to cover the operating system proper, with plenty additional for the various language translators. Although the documentation is clear and well organized, it's heavily slanted toward the high-level language users, and the assembly language programmer will find it more difficult to extract the critical information necessary to get programs running. I wasted a full day just puzzling out the fact that you have to "attach" the console input stream but "create" the console output stream. The error messages that you get along the way from the system loader are cryptic to say the least.

iRMX has been independently ported to the IBM PC by two different vendors. It is available under the name "PC/iRMX" for $2,250.00 from Real-Time Computer Science Corp. (RTCS), P.O. Box 3000, Camarillo, CA 93011. It is sold under the name "RTOS" for $600.00 by Microware, P.O. Box 79, Kingston, MA 02364. Both companies include the assembler and linker with the operating system, but charge you extra for the various high-level languages.

These two products are definitely aimed at the pro and are not for the casual user. Although it probably only took you an hour or so to read the PC-DOS manual and get comfortable with the system, it will likely take you a week to become productive with iRMX-86. After porting Laboratory Microsystems PC/FORTH to run under both the RTCS and Microware implementations, I have conluded that I like the Intel operating system and development tools very much but that the IBM PC only barely has the horsepower to make them run properly. You certainly should think twice about buying iRMX unless your PC has a hard disk and the maximum amount of RAM.

## Concerning Redirection

Chet Floyd of Manhattan Beach, California, writes: "Perhaps the good Doctor or a reader can explain a puzzling quirk of IBM's PC-DOS 2.00 in the redirection feature. File redirection works perfectly with DOS commands, but less perfectly with C and Pascal programs that use STDIN. The problem occurs when STDIN is redirected to a disk file because end-of-file is not sensed unless the EOF marker (^Z) is explicitly placed in the file. This can be easily done, of course, but having to do so is an inconvenience, particularly since DOS will hang if actual EOF occurs before the character is read. Contrary to the system documentation, ^C will not recover; the system must be rebooted.

"For example,

```
sort <text.fil
```

works swell, but

```
myprog <text.fil
```

hangs at the end of file unless ^Z is read as a character from the file.

"The IBM Pascal manual points out that STDIN never returns TRUE for EOF unless STDIN is redirected. But it seems not to work this way, and, as mentioned, C programs suffer the same way. Overcoming this problem would make the rudimentary piping and redirection that PC-DOS offers much more usable."

Any comments from the MS-DOS/PC-DOS wizards out there?

## C Programming Tools

C-INDEX+ is a new data management software tool that provides full B+Tree ISAM indexing, variable length data storage, variable length keys, and "virtual memory management" of records. The package comes with an interactive tutorial, a detailed programming guide with examples, and a reference guide by function call. C compilers currently supported include Lattice C, Microsoft C, Computer Innovations C-86, and Manx Aztec CII. There is no license fee for application software that embeds C-INDEX+. Object code license is $400.00 and source code is $2000.00 from Trio

Systems, 2210 Wilshire Blvd., Suite 289, Santa Monica, CA 90403.

Kurt Klinzing, of Novum Organum, 29 Egerton Road, Arlington, MA 02174 was kind enough to send me a review copy of his company's product called "C Building Blocks." This is a beautifully documented, incredibly comprehensive set of C function libraries. The following modules can be purchased independently:

- C Building Blocks I (string functions, printer and serial port access, directory management, file management, operating system services, and video display) $149.00.

- Advanced Building Blocks (field directed input, window management, Julian date conversion, event timers, data compression). Requires C Building Blocks I also. $99.00.

- Mathematics Building Blocks (trig, logs, exponentials, random numbers) $99.99.

- Database Building Blocks (B-Tree indexes, direct or sequential access by key, variable length records) $149.00.

- Telecommunications Building Blocks (communications port control and character I/O, modem control including autodial, file transfer with Xmodem protocol) $149.00.

The C Building Blocks are written in Lattice C and include all source code. I've embarked on a rather large application using these libraries, so a more detailed review will be forthcoming in a later column.

## Savage's Benchmark Again

This month we'll print one last source listing for a program that implements Bill Savage's floating-point benchmark, and next month we'll publish the revised, enlarged collection of results. Chris Dunford of Columbia, Maryland, writes: "I was interested in the floating-point performance/accuracy test results published in March and I thought I would see just how fast we could get an 8087-equipped IBM PC to run. The result is the attached 8088/8087 assembler program (see Listing, page 118) which executes in 2.2 seconds with an error of 3E-10 by your measure. I think this is pretty impressive. The accuracy beats everything in the March results chart except the IBM 3081, and the performance ranks tenth, ahead of such notables as HP9000, PDP-11, LSI-11, and some of the VAX timings. Here are a few notes on the program . . .

"The assembler used was version 1.3 of Digital Research's RASM-86, which supports 8087 operations (albeit with some non-standard mnemonics). Both RASM-86 and the code files produced by it run under PC-DOS. Clearly, the same results could have been achieved using the Microsoft Assembler with an appropriate set of macros; one such set is marketed by Southwestern Data Processing in Tucson. I chose RASM-86 simply on the basis of speed. Strangely, DRI's debugger, SID-86, does *not* support 8087 mnemonics even though the assembler does.

"In the interest of brevity, the program contains no code to display the final value of 'A'; a floating point-to-ASCII routine would have considerably lengthened the size of the listing (but not the execution time). I determined the 'A' value using a patched version of DEBUG which displays the 8087 register contents. Timings were obtained by adding a routine to count ticks of the system timer. This is accurate to within about 1/18th of a second, give or take a tick.

"Some readers may take exception to the fact that the algorithms employed are substantially optimized for the problem at hand. For example, there is no error checking (no errors are possible unless there is a logic flaw in the code), and there is no need for the tangent function to examine the octant of the given angle (they are all in the 45–90 degree octant). However, it's my assertion that this remains a valid comparison: one of the few remaining benefits of assembler is that such optimization is possible. That's what separates assembler code from compiler code—it's designed for the problem being solved and need not concern itself with correcting for situations that cannot possibly be encountered. In any event, I put together a version of the program with more generalized algorithms and found that the timings were substantially the same.

"I would point out that I am by no means an expert in 8087 programming, and I am certainly not a mathematician. It's quite likely that the program could be made more efficient by someone who is well-versed in either. Several of the algorithms in the program are perverted versions of those presented by Bill Rash in the Intel Application Note AP-113. I should also caution readers not to extract these algorithms for use in other programs unless they are looking forward to seeing a lot of unnormals, denormals, infinities of all shapes and sizes, and even the odd not-a-number. The routines were specifically written for this program and simply will not serve as general purpose transcendentals.

"While I was playing with all this, I also put together a test program for the p-System. The test runs under the IV.1 version as implemented by Network Consulting (version c1f). I turned off the range checking, used the 8087, and compiled to native code. Using double precision, the test ran in about 14 seconds and produced the astonishing error of 3E-10: the same error as the assembler program discussed above. My hunch is that this is artifact, but at least it is reproducible artifact. The timing, I think, is quite creditable and compares favorably with a number of the other compilers for the IBM machine. It compares extremely well with my famous brand C compiler, which took 86 seconds to produce a considerably less accurate result . . . And yes, the C compiler supports the 8087. All you 'p-System is too slow' people take note."

Thanks, Chris, for a very informative letter and program.

DDJ

**(Listing begins on page 118)**

**Reader Ballot**
Vote for your favorite feature/article.
Circle Reader Service **No. 197.**

# 16-Bit Listing (Text begins on page 116)

```
                        pagewidth 120
                        title ' DDJ Floating Point Test'
                        noiflist

                        ; ---------------------------------------------------------
                        ;                   FPTEST.A86     03/14/84
                        ; IBMPC/8087 DDJ Floating point performance/accuracy test
                        ;
                        ; By Christopher J. Dunford
                        ;    10057-2 Windstream Drive
                        ;    Columbia, Maryland 21044
                        ;    (301) 992-9371
                        ;
                        ; Portions adapted from:
                        ;   "Getting Started With the Numeric Data Processor"
                        ;   Bill Rash, Intel Applications Note AP-113
                        ;
                        ; Program performs the following C equivalent:
                        ;
                        ;   main()
                        ;   {
                        ;     unsigned i = 2499;
                        ;     double a = 1, tan(), atan(), exp(), log(), sqrt();
                        ;
                        ;     for ( ; i ; i--)
                        ;        a = tan(atan(exp(log(sqrt(a*a))))) + 1.0;
                        ;   }
                        ; ---------------------------------------------------------

                        cseg
                        main:

                        ; ----- Initialize 8088
0000 1E                         push ds                 ; Set up long ret to DOS
0001 2BC0                       sub ax,ax
0003 50                         push ax
0004 B80000         R           mov ax,data             ; Establish data addressability
0007 8ED8                       mov ds,ax
0009 B9C309                     mov cx,2499             ; Loop counter

                        ; ----- Initialize 8087 NDP
000C 90DBE3                     fninit                  ; NDP reset
000F 90D93E0000     R           fnstcw control          ; Set rounding mode to chop
0014 810E0000000C R             or control,0C00H
001A 9BD92E0000     R           fldcw control
001F 9BDB2E0400     R           fld80 half_a_pi         ; Load a constant pi/2
0024 9BDB2E0E00     R           fld80 qtr_pi            ; And a constant pi/4
0029 9BD9E8                     fld1                    ; Initialize A to 1
                        eject

                        ; ----- Main test loop begins here
                        bigloop:
002C 9BDCC8                     fmul st0,st0            ; ST = A*A
002F 9BD9FA                     fsqrt                   ; ST = sqrt(A*A)
```

*(Continued on page 120)*

# OPT-TECH SORT™

**SORT/MERGE program for IBM-PC & XT**

**Now also sorts dBASE II files!**

- Written in assembly language for **high performance**
  Example: 4,000 records of 128 bytes sorted to give key & pointer file in 30 seconds. **COMPARE!**
- Sort ascending or descending on up to nine fields
- Ten input files may be sorted or merged at one time
- Handles variable and fixed length records
- Supports all common data types
- Filesize limited only by your disk space
- Dynamically allocates memory and work files
- Output file can be full records, keys or pointers
- Can be run from keyboard or as a batch command
- Can be called as a subroutine to many languages
- Easy to use, includes on-line help feature
- Full documentation — sized like your PC manuals
- **$99** —VISA, M/C, Check, Money Order, COD, or PO
  Quantity discounts and OEM licensing available

To order or to receive additional information write or call:

**OPT-TECH DATA PROCESSING**
P.O. Box 2167 Humble, Texas 77347
(713) 454-7428
Requires DOS, 64K and One Disk Drive

Circle **no. 56** on reader service card.

# C COMPILER

- **FULL C**
- **UNIX* Ver. 7 COMPATABILITY**
- **NO ROYALTIES ON GENERATED CODE**
- **GENERATED CODE IS REENTRANT**
- **C AND ASSEMBLY SOURCE MAY BE INTERMIXED**
- **UPGRADES & SUPPORT FOR 1 YEAR**
- ***C SOURCE AVAILABLE FOR $2500.00***

| HOST | 6809 TARGET | PDP-11*/LSI-11* TARGET | 8080/(Z80) TARGET | 8088/8086 TARGET |
|---|---|---|---|---|
| FLEX*/UNIFLEX* OS-9* | $200.00 WITHOUT FLOAT<br>$350.00 WITH FLOAT | 500.00 | 500.00 | 500.00 |
| RT-11*/RSX-11* PDP-11* | 500.00 | 200.00 WITHOUT FLOAT<br>350.00 WITH FLOAT | 500.00 | 500.00 |
| CP/M* 8080/(Z80) | 500.00 | 500.00 | 200.00 WITHOUT FLOAT<br>350.00 WITH FLOAT | 500.00 |
| PCDOS*/CP/M86* 8088/8086 | 500.00 | 500.00 | 500.00 | 200.00 WITHOUT FLOAT<br>350.00 WITH FLOAT |

*PCDOS is a trademark of IBM Corp. MSDOS is a trademark of MICROSOFT. UNIX is a trademark of BELL LABS. RT-11/RSX-11/PDP-11 is a trademark of digital Equipment Corporation. FLEX/UNIFLEX is a trademark of Technical Systems consultants. CP/M and CP/M86 are trademarks of Digital Research. OS-9 is a trademark of Microware & Motorola.

**408-275-1659**

**TELECON SYSTEMS**
1155 Meridian Avenue, Suite 218
San Jose, California 95125

Circle **no. 73** on reader service card.

# HOW FAST WOULD THIS PROGRAM RUN

## IF IT WERE COMPILED USING YOUR PASCAL COMPILER ?

```
PROGRAM SIEVE;
{ THE ERATOSTHENES' SIEVE BENCHMARK }

CONST SIZE = 8190;
TYPE BYTE = 0..255;
VAR I, PRIME, K, COUNT, ITER : INTEGER;
    FLAGS : ARRAY [ 0..SIZE ] OF BOOLEAN;

BEGIN
  WRITELN( 'START' );
  FOR ITEM := 1 TO 10 DO BEGIN
    COUNT := 0;
    FOR I := 0 TO SIZE DO FLAGS[ I ] := TRUE;
    FOR I := 0 TO SIZE DO
      IF FLAGS[ I ] THEN BEGIN
        PRIME := I + I + 3;
        K := I + PRIME;
        WHILE K <= SIZE DO BEGIN
          FLAGS[ K ] := FALSE;
          K := K + PRIME
        END;
        COUNT := COUNT + 1
      END;
  END;
  WRITELN( COUNT, 'PRIMES' )
END.
```

Chances are, not as fast as it would if it were compiled using SBB Pascal.

As the following benchmarks show, SBB Pascal outperforms all other Pascal compilers for the PC in terms of speed, code size and .EXE file size:

| | Execution Time (secs) | Code Size | EXE File Size |
|---|---|---|---|
| SBB Pascal | 10.90 | 181 | 4736 |
| MS-Pascal | 11.70 | 229 | 27136 |
| Pascal/MT+ 86 | 14.70 | 294 | 10752 |
| Turbo Pascal | 15.38 | 288 | 9029 |

Development Package
$350.00

Personal Use Compiler Package
also available
$95.00

Call for free brochure with full benchmarks.

Software Building Blocks™

**607/272-2807**

Software Building Blocks, Inc.
Post Office Box 119
Ithaca, New York 14851-0119

SBB Pascal is a trademark of Software Building Blocks, Inc. MS-Pascal is a trademark of Microsoft Corporation. Pascal/MT+ 86 is a trademark of Digital Research, Inc. Turbo Pascal is a trademark of Borland International.

Circle **no. 67** on reader service card.

```
                        ; ----- Compute (ln(sqrt(A*A))) -------------------------
                        ; The NDP does all of its logs and exponentials in base 2.
                        ; Thus we must compute ln(x) indirectly using the identity
                        ;         ln(x) = ln(2) * log2(x)
                        ; Fortunately, ln(2) is available as an NDP constant.
                        ; In this and all following, 'q' is the current value of A;
                        ; at this point, q is sqrt(A*A).
0032 9BD9ED                     fldln2                  ; ST = ln(2)
0035 9BD9C9                     fxch                    ; ST = q; ST(1) = ln(2)
0038 9BD9F1                     fyl2x                   ; ST=ln(2)*log2(q) = ln(q)

                        ; ----- Compute exp(ln(sqrt(a*a))) -------------------------------
                        ; A bit tricky because the only exponential functions available are:
                        ;    y = 2^x, where x is an integer (FSCALE), and
                        ;    y = (2^x) - 1, where 0 <= x <= 0.5 (FY2XM1).
                        ; We have to split the exponent into several parts and construct the
                        ; final result from the partial results. Again, the NDP works only
                        ; in base 2, so we must use the identity:
                        ;           e^x = 2^(x*log2(e))
                        ; The algorithm also uses the identity:
                        ;           x^(y+z) = x^y * x^z
                        ; In the following, f is the fractional part and i the integral part
                        ; of q*log2(e).  See the reference for a similar routine (pp. 42-43).

003B 9BD9EA                     fldl2e                  ; Load constant log2(e)
003E 9BDEC9                     fmul                    ; ST = q*log2(e)
0041 9BD9E8                     fld1
0044 9BD9E0                     fchs                    ; ST = -1
0047 9BD9C1                     fld st1                 ; ST = q*log2(e); ST(1) = -1; ST(2) = ST
004A 9BD9FC                     frndint                 ; ST = int(q*log2(e)) = i
004D 9BD9CA                     fxch st2                ; ST = q*log2(e); ST(2) = i
0050 9BD8E2                     fsub st,st2             ; ST = frac(q*log2(e) = f
0053 9BD9FD                     fscale                  ; ST = f/2  [f*(2^-1)]
0056 9BD9F0                     f2xm1                   ; ST = 2^(f/2) - 1
0059 9BDEE1                     fsubr                   ; ST = 2^(f/2);  ST(1) = i
005C 9BDCC8                     fmul st0,st0            ; ST = 2^(f/2) * 2^(f/2) = 2^f
005F 9BD9FD                     fscale                  ; ST = 2^(q*log2(e)) = exp(q)
0062 9BDDD9                     fstp st1                ; Dump one stack level
                        eject

                        ; ----- Compute atn(exp(ln(sqrt(a*a)))) -------------------
                        ; The available function is ST = atn(ST1/ST), where
                        ; ST(1)  < ST.  This implies that we must always take the
                        ; atn of n where 0 < n < 1; however, the test requires
                        ; atn of 1 <= n <= 2499.  Help is available in the form of
                        ; atn(n) = pi/2 - atn(1/n).  We'll take the atn of 1/x and
                        ; subtract the result from pi/2.

0065 9BD9E8                     fld1                    ; ST = 1; ST(1) = q
0068 9BD9C9                     fxch                    ; ST = q; ST(1) = 1
006B 9BD9F3                     fpatan                  ; ST = atn(1/q)
006E 9BD8E2                     fsub st0,st2            ; ST = atn(1/q) - pi/2 = -atn(q)
0071 9BD9E0                     fchs                    ; ST = atn(q)
```

```
                        ; ----- Compute tan(atn(exp(ln(sqrt(a*a))))) ------------------
                        ; Too bizarre to describe here.  See the above reference, p.58ff.
                        ; The adaption is rather highly optimized for this test.

0074 9BD9F8                     fprem                   ; ST = q MOD pi/4
0077 9BD8E9                     fsubr st0,st1           ; ST = pi/4 - (q MOD pi/4)
007A 9BD9F2                     fptan                   ; Compute partial tangent
007D 9BDEF1                     fdivr                   ; Compute final tangent

                        ; ----- End of loop.  Add 1 to A and loop back
0080 9BD9E8                     fld1                    ; ST = 1;  ST(1) = A
0083 9BDEC1                     fadd                    ; ST = A + 1
0086 E2A4           002C        loop bigloop
0088 CB                         retf

                        ; ----- Data area ---------------------------------------------
                        dseg word
0000                    control   rw 1                                  ; NDP control
0002                    status    rw 1                                  ; NDP status
0004 35C26821A2DA       half_a_pi dw 0C235H, 02168H, 0DAA2H, 0C90FH, 03FFFH     ; 80-bit pi/2
     0FC9FF3F
000E 35C26821A2DA       qtr_pi    dw 0C235H, 02168H, 0DAA2H, 0C90FH, 03FFEH     ; 80-bit pi/4
     0FC9FE3F
                        end main

END OF ASSEMBLY.  NUMBER OF ERRORS:  0.  USE FACTOR:  0%
```

End Listing

Michael Wiesenberg

## Give Me Your Phone Number

There are probably lots of vendors who wonder each month why their particular product or service goes unnoticed by this column. Let me explain in part how material for this column is filtered. *DDJ* receives literally hundreds of press releases each month. Before I even see the stack, someone usually culls announcements of the appointment of Sharon Apartment as Vice President of Marketing of Plastic Fantastic Diskettes and of the move of Hitek Analytic Engines from Palo Alto into their newly expanded facilities in Milpitas. I sift through the remaining dross for gems that I hope will be "Of Interest."

I almost automatically reject announcements of products unaccompanied by price because I figure if a company doesn't want to let people know how much its widget costs before a prospective customer sends an inquiry, the product must be overpriced. I also rarely use a product description sent by a company that doesn't list its phone number. I'm trying to look out for the readers, and I think a company that doesn't want phone calls has no intention of supporting its products. If I receive a press release that looks as if it were written by a grade school dropout, I also usually pitch it, my reasoning being that a company that can't take care in writing press releases probably does indecipherable documentation and is likely to take less care in the design of its products.

So there you have it. Maybe my conclusions are unwarranted, but that's how I operate. Companies that want attention should give me a phone number and tell me how much the product costs. They shouldn't attempt to hide the product's deficiencies behind a lot of overblown puffery. (I'm a technical writer by trade; I see through that stuff.) And if they can't write decent English, they should hire someone who can to write the release.

## Don't Be Embarrassed in Spanish

**El Ortografico**, from Ibersoft (good name!), is the first-ever spelling checker for the Spanish language. It also checks for proper accents and lets you look up on-line the conjugation of most verbs. It also verifies your corrections as you enter them. You can use this speller with any word processor that stores text in ASCII, and you can customize for your word processor the unique characters of the language. If your word processor cannot represent the special characters, El Ortografico also has a utility program that permits printing them (if your printer is able to do so). The program runs on TRS-80 Models I, III, and 4, CP/M, PC-DOS, and MS-DOS, and will soon be available for Apple II. Sounds like a great product for $99.95. **Reader Service No. 101.**

## Spell 1000 Times Faster

Since there seems to be interest in TEX and since it appears that a number of you have HP computers or use them on the job, I offer the following. JDJ Wordware offers **JSPEL/1000** for RTE-6/VM and RTE-A operating systems on HP 1000 computers. It features high-speed performance (in excess of 30,000 words per minute) by word caching. It generates almost instantaneously up to 10 correction candidates for what it considers an error. You merely enter the number of the correction you wish, and JSPEL makes the replacement. Each error is displayed in place in the line where it occurs, and you can generate a 20-line window of context around the suspect word by typing one letter.

JSPEL uses multiple dictionaries: a 27,000-word main dictionary that you may expand indefinitely, an incremental dictionary to store new words you wish to add to the main dictionary, and a customized dictionary that can be specific to a particular file—containing, for example, computer acronyms for a document about computers. JSPEL detects adjacent repeated words words. (It would have corrected that last sentence for me.) It can be used in look-up mode to find any words, and the search patterns can include any arbitrary selection of several kinds of wild cards. You TEX users will be pleased to know that JSPEL ignores TEX code sequences. The program supports the new hierarchical file system. A relocatable license costs $625, and source is $1500. You then can get software support for $20 a month and manual update service for $5 a month. The reference manual alone is $15. **Reader Service No. 103.**

## TEXies Meet

The **TEX User's Group** will meet at Stanford August 13–24, during which two courses will be offered in the use of TEX: *Book Design Utilizing TEX* on the 13th and 14th, and *TEX for Beginners* August 20–24. Write to TUG care of the American Mathematical Society. **Reader Service No. 105.**

## Plastic Seat Covers for Keyboards

Have you seen those tacky plastic seat covers people use in their homes when they don't want anyone to get their precious couches and chairs dirty? Now you can get them for your com-

puter's keyboard, in case you're the kind likely to spill coffee on it. **Safeskin** is molded to fit a particular keyboard like a glove and to remain in place during use. It has tactile "homerow" and numeric character locators and is made of antistatic polymer, through which you can see easily key tops and side markings. Merritt Computer Products sells them for IBM PC and compatibles, Apple IIe, TI Professional, TRS-80, Televideo, and Wang keyboards; they cost $29.95. **Reader Service No. 107.**

## Z-100 Memory and Storage

PIICEON has a 256K memory board for Zenith Z-100 at $750; a 10Mb Winchester drive, the **ZD 100-10**, for $1795; and a 20Mb drive, the **ZD 100-20**, for $2285. All come with software and all necessary cables and connectors and are compatible with all levels of Z-DOS. PIICEON also has memory and disk upgrade kits for IBM PC and Alpha Microsystems. **Reader Service No. 109.**

## Hang a PC on your Model 100

If you own a Radio Shack Model 100 computer (and sales figures for the little portable indicate that it can't be *just* computer-magazine writers who have been buying it), you might be interested in a program that lets the Model 100 use the disk storage of another computer. **Disk+** from the Portable Computer Support Group in Dallas is supplied in two pieces—part on cassette for the Model 100 and part on disk for the other computer: currently, the IBM PC and most MS-DOS machines, Radio Shack computers, Apple II, IIe, II+, Olivetti ETV 300, M20 and M24, and some CP/M computers. You select Disk+ from your Model 100 RAM file menu, and the menu turns into a disk menu showing the files on your other machine. You can then pull files into Model 100 RAM from disk or save RAM files to disk at up to 19200 baud; your large computer has become a peripheral for your Model 100. You can also create and manipulate disk subdirectories to handle groups of files at one time. Disk+ costs $69.95 and requires a serial cable with a null modem (unless you use the Model 100's internal modem; not recommended, since you'll be limited to 300 baud disk accesses). One caveat: this thing uses 9K of the Model 100's limited RAM. **Reader Service No. 111.**

## Your China Trip

If you're planning to be in Xiamen toward the end of November you should check out **Computer China '84**, an exhibition introducing the latest micro and minicomputer technology to the people of the People's Republic. It seems that the personal computer revolution is catching fire in China, with grass-roots microcomputer associations and user's groups springing up in the provinces. There are only about 40,000 micros in China today, including single-board computers, but Adsale Exhibition Services, which is coordinating overseas vendors and exhibitors, expects a good turnout for the exposition-plus-conference. **Reader Service No. 113.**

## Price Cut at the Co-op

**CPMUG**, the CP/M User's Group, has cut the prices for its volumes of CP/M software. In case you were unaware, CPMUG's been running a huge software exchange program for some time now, some 100 volumes of CP/M software of mixed quality, including everything from source-code interpreters and compilers to games and printer pictures. They sell it in bulk, like wheat germ at the co-op, on 8-inch IBM or 5¼-inch Kaypro double-density single-sided, Epson QX-10 double-density double-sided, Apple 16-sector, and North Star double- or quad-density

disks. They cut the price from $13 to $10 for 8-inch disks and from $18 to $15 for 5¼-inch disks; that's for the U.S., Canada, and Mexico. They dropped the corresponding prices for other destinations from $17 to $13 and from $21 to $18, respectively. CPMUG will send you a catalog for $10 ($15 for you non-North Americans). **Reader Service No. 115.**

## CP/M Calling CP/M

**Softcom Telecommunications Utility** for CP/M, from The Software Store, is a terminal emulator for mainframe time-sharing systems; it downloads files from a host system, sends text from your disk to, they claim, "almost any type of computer," and exchanges any type of file with other Softcom systems. The intelligent terminal mode transfers data at up to 9600 baud in full or half duplex and supports the XON/XOFF protocol. You need an 8080, 8085, or Z80 CP/M system, with at least 32K, and $150. **Reader Service No. 117.**

## Half a Board Is Better Than . . .

One of the first in the race for add-ons for the IBM PC Portable is Ven-Tel with their **PC Modem Half Card**, a 1200/300 baud, auto-answer, auto-dial internal modem, for $549. Since the Portable has only half-sized expansion slots, only boards of this size fit inside the computer. Apparently modified from their Half Card modem for the PC-XT, the Half Card comes with CrossTalk-XVI, instructions, and a phone cable. **Reader Service No. 119.**

## Hinkey Dinkey Disk-a-Do

If you have diskettes scattered all over the top of your desk gathering dust and soaking up electric fields, you need **Disk-a-Do** diskette storage units from Information Concepts. These rotating lazy-Susan units are made of molded black ABS *(what's that?)* supported on a steel base plate with ball bearings. They have stenciled slot index numbers, and the first slot has a printed index directory card. Model 60 holds 60 diskettes and costs $74.95. Model 60C holds 60 and has a bronze acrylic dust cover cabinet with a window toward which you rotate the unit until the diskette you want appears; it costs $99.95. There's also a 120C that holds 120 diskettes and has the cover, but they neglected to tell us the price. Add $2 p&h per unit. **Reader Service No. 121.**

## Where's the Widower?

I'm waiting for a lot of irate letters from feminists on this one. The **Computer Widow Tee Shirt** from Crabapple is, they say, "an ideal gift for that wife or girl friend who thinks you spend too much time computing. [It] features white high tech lettering on an all black fabric and is a perfect way to compensate for all those evenings you spent at the keyboard." $9.95 plus $1 p&h. **Reader Service No. 123.**

## IBM Recalls Everything (Except IBM)

IBM (Incredibly Big Manufacturer) has announced, through its wholly owned subsidiary, the United States Government, a recall on all privately held personal computers, citing a defective connector that could disintegrate during normal operation, releasing deadly dimethyltryptamine gas. (Bonuses will be paid those turning in plug-compatible look-alikes.) Recalled computers will be replaced at no cost with IBM PCjrs that have been modified to accept only IBM software. (All so-called compatible software will self-destruct in these special machines.) Naturally these jrs are compatible with no other IBM products (or anything else). In other news, IBM is rumored to be planning a midyear four-fold increase in the prices of all PCjr software. **Reader Service No. 3.141592654.**

## Contact Points

Adsale Exhibition Services, 21/F, Tung Wai Commercial Building, 109-111 Gloucester Road, Wanchai, Hong Kong; 5-8920511 (phone); 63109 ADSAP HX (Telex).

American Mathematical Society, Box 6248, Providence, RI 02940; (401) 272-9500, ext. 232.

Ariel Corporation, 600 West 116th St., New York, NY 10027; (212) 662-7324.

CP/M User's Group (CPMUG), 1651 Third Avenue, New York, NY 10028.

Crabapple, Inc., Box 3236, Framingham, MA 01701; (617) 877-9242.

Ibersoft, Box 3343, Trenton, NJ 08619; (609) 890-1496.

Information Concepts, Inc., Box 462, Stone Mountain, GA 30086; (404) 979-8479.

JDJ Wordware, Box 354, Cupertino, CA 95015; (415) 965-3245.

Merritt Computer Products, Inc., 2925 LBJ Fwy., Suite 180, Dallas, TX 75234; (214) 942-1142.

PIICEON, 2114 Ringwood Ave., San Jose, CA 95131; (408) 946-8030.

Portable Computer Support Group, 11035 Harry Hines Blvd., #207, Dallas, TX 75229; (214) 351-0564.

TEX User's Group: see American Mathematical Society.

The Software Store, 706 Chippewa Square, Marquette, MI 49855; (906) 228-7622.

Ven-Tel, Inc., 2342 Walsh Ave., Santa Clara, CA 95051; (408) 727-5721.

DDJ

# ADVERTISER INDEX

| Reader Service No. | Advertiser | Page No. |
|---|---|---|
| 1 | 2500AD Software | 64 |
| 2 | AccuData Software | 123 |
| 3 | Amber Systems | 75 |
| 4 | Application Executive Corp | 100 |
| 5 | Ashton-Tate | 4 |
| 6 | Ashton-Tate | 9 |
| 7 | Avocet Systems, Inc. | 55 |
| 8 | BD Software | 75 |
| 9 | B.G. Micro | 113 |
| 10 | Bonnie Blue Software | 49 |
| 11 | Borland International | 128 |
| 12 | Byte Show | 24 |
| 13 | Carousel Micro Tool | 16 |
| 14 | The Code Works | 83 |
| 15 | Compu-Draw | 126 |
| 16 | Compusophic Systems | 12 |
| 17 | Computer Friends | 104 |
| 18 | The Computer Journal | 125 |
| 19 | Coriolis Company | 125 |
| 20 | Creative Solutions | 97 |
| 21 | C User's Group | 35 |
| 22 | C Ware | 71 |
| 23 | Data Access Corporation | 59 |
| 24 | Datalight | 79 |
| 25 | Dedicated Microsystems | 109 |
| 26 | D & W Digital | 45 |
| 27 | Ecosoft, Inc. | 11 |
| 28 | Foehn Consulting | 125 |
| 29 | GGM Systems, Inc. | 109 |
| 30 | GTEK | 79 |
| 31 | Hallock Systems Consultants | 114 |
| 32 | Harvard Softworks | 87 |
| 33 | Integral Quality | 111 |
| 34 | Key Solutions | 22 |
| 35 | Laboratory Microsystems | 71 |
| 36 | Lattice, Inc. | 121 |
| 37 | Leo Electronics, Inc. | 79 |
| 38 | Lifeboat Associates | 51 |
| 39 | Logical Devices | 16 |
| 40 | Logical Systems | 75 |
| 41 | MicroMotion | 107 |
| 42 | Microprocessors Unlimited | 125 |
| 43 | National Software Show | 39 |
| 44 | Next Generation Systems | 50 |
| 45 | OCCO, Inc. | 21 |
| 56 | Opt-Tech Data Processing | 119 |
| 46 | Overbeek Enterprises | 113 |
| 47 | Phlexible Data Systems | 22 |
| 48 | Phoenix Software | 11 |
| 81 | Phoenix Software | 13 |
| 82 | Phoenix Software | 15 |
| 49 | Port-A-Soft | 100 |
| 50 | ProCode International | 111 |
| 51 | The Programmer's Shop | 67 |
| 52 | Solutions Systems | 67 |
| 53 | Solutions Systems | 67 |
| 54 | PRO Microsystems | 83 |
| 55 | Protools | 125 |
| 57 | Quest Research | 127 |
| 58 | Quick-N-Easi Products, Inc. | 32 |
| 59 | Rational Systems, Inc. | 113 |
| * | Edward Ream | 109 |
| 60 | The Redding Group | 100 |
| 61 | Revasco | 107 |
| 62 | Sage Computer Technology | 2 |
| 63 | SemiDisk Systems | 19 |
| 64 | Shaw Laboratories | 90 |
| 65 | Simpliway Products Company | 125 |
| 66 | SLR Systems | 83 |
| 67 | Software Building Blocks | 119 |
| 68 | Software Engineering Consultants | 107 |
| 69 | Software Horizons, Inc. | 14 |
| 80 | The Source View Corporation | 91 |
| 70 | Southern Computer Corporation | 71 |
| 71 | Syntax Constructs | 125 |
| 72 | Systems Guild, Inc. | 125 |
| 73 | Telecon Systems | 119 |
| 74 | Unified Software Systems | 121 |
| * | Frank N. Vitaljic | 125 |
| 75 | Mark Williams & Company | 3 |
| 76 | Wordtech Systems, Inc. | 29 |
| 77 | Workman & Associates | 126 |
| 78 | *DDJ* Change of Address | 92 |
| 79 | *DDJ* Advertise | 35 |

# COHERENT™ IS SUPERIOR TO UNIX* AND IT'S AVAILABLE TODAY ON THE IBM PC.

Mark Williams Company hasn't just taken a mini-computer operating system, like UNIX, and ported it to the PC. We wrote COHERENT ourselves. We were able to bring UNIX capability to the PC with the PC in mind, making it the most efficient personal computer work station available at an unbelievable price.

For the first time you get a multi-user, multitasking operating system on your IBM PC. Because COHERENT is UNIX-compatible, UNIX software will run on the PC under COHERENT.

The software system includes a C-compiler and over 100 utilities, all for $500. Similar environments cost thousands more.

COHERENT on the IBM PC requires a hard disk and 256K memory. It's available on the IBM XT, and Tecmar, Davong and Corvus hard disks.

Available now. For additional information, call or write,

Mark Williams Company
1430 West Wrightwood, Chicago, Illinois 60614
312/472-6659


COHERENT is a trade mark of Mark Williams Company.
*UNIX is a trade mark of Bell Laboratories.

# dBASE III™


dBASE III™
SYSTEM DISK
Serial No: 100001
Format: IBM PC, XT and 100% compatibles
PC-DOS 2.0
Version: 1.0
ASHTON-TATE
10150 West Jefferson Blvd.
Culver City CA 90230 (213) 204-5570
TELEX 664228 ASHTATE CULV
©Copyright 1984 Ashton-Tate
as an unpublished licensed
proprietary work.
All rights reserved.